A Comparative Study on Secure Network Virtualization

Serdar Cabuk, Chris I. Dalton, Aled Edwards, Anna Fischer
HP Laboratories
HPL-2008-57
May 21, 2008*

virtualization, security, networking

Secure and efficient network virtualization is a key building block for virtualized environments such as data centers or enterprise networks. While machine virtualization alone provides immediate isolation of computing resources such as memory and CPU between guest domains, the network remains to be a shared resource as all traffic from guests eventually pass through a shared network resource and end up on the shared physical medium. As a result, we need mechanisms to (1) control the information flow between virtual machines (e.g., who can communicate with whom), (2) configure virtual and physical network resources, and (3) separate network resources used by each networking domain. Within HP Labs we have successfully developed and deployed technologies that enable secure networking within virtualized infrastructures. In this report, we present the findings of a comparative study that we conducted to evaluate the security, performance, and manageability of these approaches. We further report our experiences with prototype implementations on Xen platforms.

External Accession Date Only

Approved for External Publication

A Comparative Study on Secure Network Virtualization

Serdar Cabuk, Chris I. Dalton, Aled Edwards, Anna Fischer
Hewlett-Packard Laboratories
Bristol, United Kingdom
{serdar.cabuk, cid, aled.edwards, anna.fischer}@hp.com

ABSTRACT

Secure and efficient network virtualization is a key building block for virtualized environments such as data centers or enterprise networks. While machine virtualization alone provides immediate isolation of computing resources such as memory and CPU between guest domains, the network remains to be a shared resource as all traffic from guests eventually pass through a shared network resource and end up on the shared physical medium. As a result, we need mechanisms to (1) control the information flow between virtual machines (e.g., who can communicate with whom), (2) configure virtual and physical network resources, and (3) separate network resources used by each networking domain. Within HP Labs we have successfully developed and deployed technologies that enable secure networking within virtualized infrastructures. In this report, we present the findings of a comparative study that we conducted to evaluate the security, performance, and manageability of these approaches. We further report our experiences with prototype implementations on Xen platforms.

1. OUTLINE

This report is organized as follows: In Section 2, we list the functional and security requirements for secure network virtualization in data centers. In Section 3, we present three networking designs that are currently under development in HP Labs. We present the findings of our comparative study in Section 4. In Section 5, we report our experiences with two prototype implementations on Xen platforms. We present related work in Section 6 and conclude in Section 7.

2. AIMS AND REQUIREMENTS

Secure network virtualization allows parties to define virtual topologies on top of physical infrastructure in a secure manner. In this section, we list the networking, security, and performance requirements for enabling and securing network virtualization on virtualized data centers.

2.1 Networking Aims

Machine virtualization allows a single physical machine to run several (perhaps different) operating systems concurrently creating the illusion of multiple machines each running their own operating system. In a similar fashion, network virtualization allows groups of related virtual machines running on separate physical machines to be connected together as though they were on their own separate network fabric. Virtual network extensions further enable the creation of arbitrary virtual network topologies in virtualized infrastructures independent of the underlying physical network topology. As an example, Figure 1 illustrates how a simple computer infrastructure consisting of two machines connected together via a single physical LAN segment can be mapped to virtual machines and a virtual LAN segment. In this setting, physical machine A hosts two virtual machines. One of them (A2) is connected into virtual LAN segment 1. Physical machine B also hosts two virtual machines. One of them (B1) is also connected into virtual LAN segment 1. Virtual machine A2 and Virtual machine B1 appear as though they are connected together via a single LAN segment even though in reality the physical network connecting them consists of multiple LAN segments interconnected via routers.

Virtual machines C2 and D1 are also connected together via virtual LAN segment 2. Traffic is isolated between virtual LAN segments so machines C2 and D1 cannot see the traffic passing between A2 and B1 even though they are sharing the underlying physical network infrastructure. Figure 2 shows a more sophisticated topology, this time consisting of two LAN segments interconnected by a routing device. It also shows a possible instantiation of this using virtual network extensions.

In some circumstances, it may be that the virtual topology is much simpler than the physical topology. This is the case in the first example where two machines appear as though they are connected directly together on the same LAN segment even though in reality they may be separated by many networking components, including potentially being at opposite ends of a Wide Area Network (WAN) link. On the other hand, as in the second example, it may be desirable that a particular virtual topology is more complicated than the underlying physical topology. For example, network segmentation (such as a DMZ arrangement) is often used for reasons of security to protect services running on machines connected to that network. It should be possible to offer those same kind of security properties and protections to the services hosted on virtual machines by the use of segmented virtual networks, irrespective of the underlying physical network arrangement.

Of course, isolated virtual networks just linking virtual machines with each other are of limited use. We also want to allow interaction with traditional non-virtualized entities anywhere on the Internet, for example, to offer third-party hosts occasionally to be connected to the datacenter environment. Therefore, virtual network extensions must also allow for bridging between the virtual and the physical network.

2.2 Security Requirements

We describe the security objectives for network virtualization through security policies that define confidentiality, integrity, information flow control, and isolation requirements.

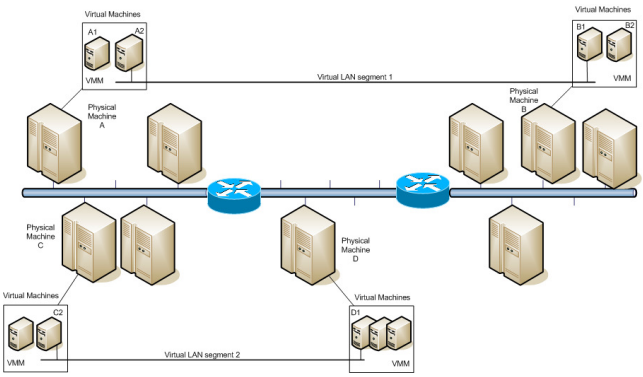


Figure 1: Virtual to Physical network mapping.

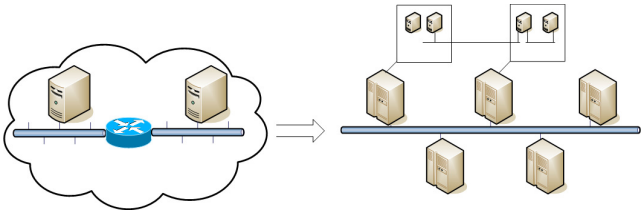


Figure 2: Physical mapping of router connected virtual LAN segments.

In our work, we treat each virtual LAN segment as a security domain with necessary mechanisms in place that enforce the aforementioned security policies within domains (intra-domain policies) as well as between domains (inter-domain policies). In particular, the confidentiality requirement is geared towards the separation of traffic that originates from different virtual LAN segments. This separation is essential because in most cases the physical infrastructure is shared among various virtual LAN segments. In cases where the physical infrastructure is trusted (e.g., in a controlled datacenter) no additonal scheme is required. In other cases (e.g., on a WAN link) appropriate measures need to be taken to ensure the confidentiality inline with each domain policy.

In addition to the confidentiality requirement, a security domain must be able to enforce intra-domain policies that determine the conditions domain entities need to adhere to prior to being admitted as members. In virtual networking speak, all entities that request membership to the particular virtual LAN segment should behave as expected – i.e., as defined by the segment policy. A straightforward way to ensure this is to verify the hardware and software configuration of the prospective members either statically prior to joining or dynamically in an ongoing fashion.

The traffic flow between security domains may also be subject to strict controls as defined by inter-domain policies. These bidirectional flow control policies determine which hosts belonging to different security domains are allowed to communicate and under what conditions. These policies are usually enforced at the security domain perimeter by trusted entities with hardened configurations. One example is a trusted dual-homed host that connects two security domains while enforcing traffic flow control policies (e.g., firewall rules that determine the traffic types that are allowed to be forwarded). As we illustrate in Figure 3, this host uses two physical NICs (one for each domain) and enforces the inter-domain flow policy for the information that flows between the NICs.

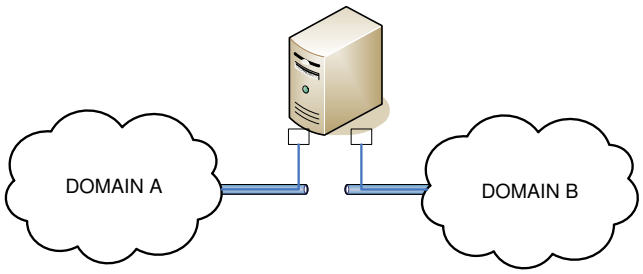


Figure 3: An example dual-homed host with two NICs for inter-domain flow control between two security domains.

Another inter-domain requirement is the isolation of virtual resources used by each security domain. In particular, isolation refers to the requirement that resources used by two security domains are logically separated and there is no unintended direct information flow using such logical resources[1]. Note that resource-based flow control is different from the aforementioned traffic-based flow control between security domains. The former flow control type prevents the misuse of shared resources to leak information (e.g., listening to virtual NICs of other virtual machines on a physical host) whereas the latter controls the network traffic between domains.

2.3 Performance Requirements

Virtual networking is central to virtualized data center operations. Hence, the added network virtualization logic should match the required operational levels of the datacenter. Poorly performing virtualized networking components can potentially affect all datacenter operations ranging from VM deployment to inter-VM communications. Additionally, the logic should be scalable to support the sheer number of VMs in a virtualized data center. Certain availability levels should be guaranteed for virtual networking components (e.g., virtual switches) lack of which can potentially jeopardize data center operations and lead to Denial of Service (DOS) attacks. Lastly, it is important to isolate and localize performance requirements to each security domain. With such guarantees in place, each domain can operate independently in concordance with the Service Level Agreement (SLAs) as required by the domain owner (i.e., a data center customer). These SLAs may define availability requirements for the underlying virtualized network infrastructure, but may also include network bandwidth guarantees or limitations, so that data center customers that require high bandwidth communication can be charged more than customers that are running low bandwidth applications. Virtual network extensions need to provide mechanisms to control virtual network performance in a secure and reliable fashion.

Eventually, it is desired that entities of different security domains should be allowed to talk to each other. In this

[1]Indirect communication flows such as covert channels are outside the scope of this paper.

case the solution ideally has to take into account the performance requirements of all involved domains. Also, it is important that inter-domain communication does not create any bottlenecks in the infrastructure – which is a challenge when using an entity that becomes a single point of control for cross-domain communication (e.g., the dual-homed host illustrated in Figure 3). Overall, it is important that an analysis of performance requirements considers the actual usage model of the system. However, one can argue that in many scenarios most network traffic will be within a single domain rather than across domain boundaries, and in these cases a slight decrease in inter-domain performance might be acceptable to ensure better security and manageability of the solution.

Added network virtualization logic imposes a certain level of overhead on datacenter operations. In this paper, we empirically assess the overhead of each network virtualization technology in comparison to each other and to the non-virtualized equivalent. We additionally investigate the availability, scalability, and isolation (in relation to performance) properties that can be guaranteed by each technology.

3. NETWORK VIRTUALIZATION

Network virtualization abstracts away the underlying network infrastructure and topology. In this section, we present various schemes we have designed that enable this abstraction in a security preserving way. In Sections 3.1 and 3.2, we introduce the virtual switch approach that employs EtherIP and VLAN tagging, respectively. In Section 3.3, we present MAC rewriting that follows a different virtualization approach from the former two. We highlight the differences between the three approaches in Section 4.

3.1 Virtual Switch and EtherIP

One option for virtual networking is to virtualize at the IP level. However, to avoid problems with supporting non-IP protocols and IP support services such as ARP that sit directly on top of the Ethernet protocol, we chose to virtualize at the Ethernet level. Figure 4 shows the main abstraction of our virtual network extensions in which we divide a virtual network into virtual LAN segments. For our purposes, a LAN segment is considered to be an Ethernet broadcast domain.

3.1.1 Networking Design

Each virtual LAN segment is represented by a *virtual switch*. A virtual machine appears on a particular virtual LAN if one of its (virtual) network interface devices is "plugged" into one of the switch ports on the virtual switch forming that segment. The virtual switch behaves like a normal physical switch. Ethernet broadcast traffic generated by a virtual machine connected to the switch is passed to all virtual machines connected to that switch. The virtual switch builds up a forwarding table based on observed traffic so that non-broadcast Ethernet traffic can be delivered in a point-to-point fashion, as in a real switch.

The virtual switch is designed to operate in a distributed fashion. The privileged domain (or host OS, depending on the virtualization technology in use) on each physical machine managing a VM connected to a particular virtual LAN segment, runs part of the virtual switch forming that virtual LAN segment. A component of the privileged domain captures the Ethernet frames coming out of a VM virtual network device. This component is configured to know which particular virtual switch the VM is supposed to be connected to.

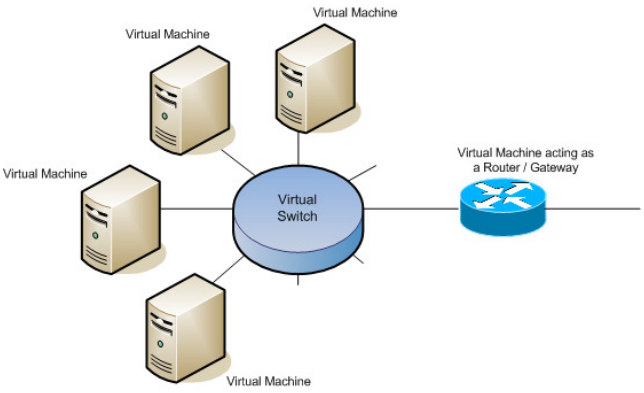


Figure 4: Abstract view of the Virtual Network Extensions.

EtherIP encapsulation. The VM Ethernet frames are encapsulated in IP packets using EtherIP. EtherIP is a standard protocol for tunneling Ethernet and 802.3 packets via IP datagrams and can be employed to expand a LAN over a Wide or Metropolitan Area Network [12]. In this setting, each tunnel endpoint uses a special network device provided by the operating system that encapsulates outgoing Ethernet/802.3 packets in new IP packets. We insert virtual LAN membership information (i.e., the virtual LAN identifier, which is unique for each virtual LAN segment within a virtual network) into the EtherIP header of each encapsulated packet. The encapsulated packets are then transmitted to the other side of the tunnel where the embedded Ethernet/802.3 packets are extracted and transmitted to the destination host that belongs to the same virtual LAN segment.

Address mapping. The virtual switch component on a VMM maps the Ethernet address of the encapsulated Ethernet frame to an appropriate IP address. This way, the encapsulated Ethernet frame can be transmitted over the underlying physical network to physical machines hosting other VMs connected to the same LAN segment that would have seen that Ethernet traffic had the VMs actually been on a real LAN together. The IP address chosen to route the encapsulated Ethernet frames over the underlying physical network depends upon whether the encapsulated Ethernet frame is an Ethernet broadcast frame or not and also whether the virtual switch has built up a table of the locations of the physical machines hosting other VMs on a particular LAN segment based on observing traffic on that LAN.

Broadcast / multicast mapping. IP packets encapsulating broadcast and multicast Ethernet frames are given a multicast IP address and sent out over the physical network. Each virtual LAN segment has an IP multicast address associated with it. All the physical machines hosting VMs on a particular virtual LAN segment are members of the multicast group for that virtual LAN segment. This mechanism ensures that all VMs on a particular virtual LAN segment receive all broadcast Ethernet frames from other VMs on that segment.

Non-broadcast mapping. Encapsulated Ethernet frames that contain a directed Ethernet address destination are either flooded to all the VMs on a particular LAN segment (using the IP multicast address as in the broadcast case) or sent to a specific physical machine IP address. This depends upon whether the virtual switch component on the encapsulating VM has learned the location of the physical machine hosting the VM with the given Ethernet destination address based on traffic observation through the virtual switch.

3.1.2 Requirements Revisited

Layering over arbitrary physical networks. The decision to encapsulate Ethernet frames from VMs within IP packets allows us to connect different VMs to the same virtual LAN segment as long as the physical machines hosting those VMs have some form of IP based connectivity between them. This also enables the connection of virtual network segments over a WAN link, even though that requires some form of tunnelling and appropriate multicast support - which is not always a given. In general there are no restrictions on the topology of the underlying physical network, meaning it could be a fully switched network or have any kind of routed configuration.

Routing within virtual networks. Currently, a router within a virtual network is provided by the use of virtual machine with multiple virtual network interface cards. The interface cards are plugged into ports on the different virtual switches that it is required to route between. Standard routing software is then configured and run on the virtual machine to provide the desired routing services between the connected LAN segments.

Gatewaying to non-virtualized systems. To allow for communication with systems that live in the non-virtualized world we provide a gateway. The gateway is simply a virtual machine with two virtual network interface cards. One of the cards is plugged into a port on a virtual switch. The other virtual network card is bridged directly on to the physical network. The gateway has two main roles. First, it advertises routing information about the virtual network behind it so that hosts in the non-virtualized world can locate the virtual machines residing on a virtual network. Second, the gateway converts packets to and from the encapsulated format required of our virtual networks.

3.2 Virtual Switch and VLAN Tagging

VLAN Tagging is a well-established network virtualization standard for isolation on physical network equipment. The standard is described in IEEE 802.1Q and uses tagging of Ethernet packets for isolation between networks [11]. VLAN tagging is used in non-virtualized environments where, e.g., a host in the VLAN 42 uses a special network device provided by the operating system to apply a VLAN tag, which contains the VLAN ID 42, to outgoing packets and remove the tag from incoming packets before they are processed by the upper network stack. VLAN-capable physical switches ensure that by default packets only flow within VLANs.

3.2.1 Networking Design

We employ VLAN tagging as an alternative to EtherIP encapsulation for efficiency purposes. As an example, in a controlled data center environment where WAN connectivity may not be required, the use of efficient VLAN-enabled switches that provide sufficient isolation on the wire yields performance gains over EtherIP encapsulation as we report in Section 5. In this case, VMs are assigned to one or more VLAN(s), and each VLAN segment employs its own VLAN-capable virtual switch module to tag Ethernet frames. This module resides within the host OS or privileged domain that facilitates the networking capabilities, captures packets coming from VMs and tags those with the ID of the VM's VLAN before sending them onto the physical wire. On the receiving side, the module removes the VLAN tag and passes the packets untagged into the destination VM(s). Packets are only tagged when they have to be transmitted over the physical network. VMs are unaware of the VLAN tagging and send/receive packets without any VLAN information.

To handle VLAN tagged packets, the physical network equipment needs to support IEEE 802.1Q and be configured accordingly. As an example, if a machine hosts a VM that is part of VLAN 42, then the switch port that is used by that machine needs to be assigned to that specific VLAN 42. Of course, a machine might host multiple VMs which can be in different VLANs, and therefore a switch port might be assigned to multiple VLANs (which creates a `VLAN trunk` between the host and the switch port). Whenever a host deploys a new VM or removes a VM, the switch port might need to be reconfigured. Ideally, this can be done in a dynamic and automated fashion, e.g., through network management protocols such as, for example, the GARP VLAN Registration Protocol (GVRP). As the physical switches only pass packets between machines within the same VLAN, those provide an additional isolation mechanism to our VLAN-capable virtual switch that is deployed on all of the hosts.

Address mapping. In contrast to both the encapsulation and the MAC Rewriting approach, the VLAN tagging solution does not require any address mapping mechanism. VMs discover address information of other VMs by using standard discovery protocols in the same way as in a non-virtualized environment. However, the virtual switch module that runs on each physical machine learns Ethernet addresses attached to the virtual switch ports by inspecting packets (in the same way as physical switches do) and builds up lookup tables (one table per virtual switch / VLAN) that store information about the location of VMs based on their Ethernet addresses. The virtual switch uses this table to decide if a packet has to be passed to a local VM or onto the physical network to be delivered to a remote machine.

Broadcast / multicast mapping. There is no explicit mapping of broadcast / multicast addresses as in the case of EtherIP encapsulation or MAC Rewriting. Instead, physical switches that manage the underlying network infrastructure ensure that broadcast and multicast traffic never cross VLAN boundaries. Broadcast and multicast packets that are tagged with a VLAN ID are passed to all switch ports that are associated with that particular VLAN, but no other ports. When those packets enter the physical machine that runs our virtual switch module, the packets are only passed into VMs that are attached to virtual switch ports that are assigned to the VLAN matching the ID in the packets.

3.2.2 Requirements Revisited

Layering over arbitrary physical networks. The VLAN tagging solution can be used over arbitrary layer 3 networks. However, in contrast to the encapsulation ap-

proach, a solution based on pure VLAN tagging is limited to a LAN environment and cannot be deployed over WAN links. VLAN tagging is highly dependent on support from the physical network equipment that is managing the underlying infrastructure. E.g., switches need to support the VLAN tagging standard that we use when tagging our packets in our virtual switch module (IEEE 802.1Q) and need to be configured to handle tagged packets in order to provide appropriate isolation between VLANs.

Routing within virtual networks. A VLAN is a logical network segment and by default network traffic such as broadcast messages or ARP communication is limited to a single VLAN. However, it is also possible to allow communication between (virtual) machines of different VLANs through `Inter-VLAN routing`. There are multiple well-known and standardized solutions to allow this. For example, most of today's network switches facilitate fast layer 3 routing between multiple VLANs. This solution offers high performance, but requires that routing policies can be configured on the physical network devices – ideally in an automated fashion. As an alternative, we can also deploy specific VMs that have multiple network interfaces in multiple VLANs and route packets between those – as in the EtherIP encapsulation approach.

Gatewaying to non-virtualized systems. In VLAN tagging, communication with non-virtualized systems is straightforward. This is because VLAN tagging is a widely used standard that is deployed within infrastructures where physical machines do not run any (network) virtualization software. There is no need for a gateway VM as in the EtherIP encapsulation approach. Instead, physical switches can be configured to remove VLAN tags from packets when transmitting on a port where the connected endpoint is not VLAN-capable, and add tags whenever packets are received on that specific port. In that case those endpoints are completely unaware of VLANs, and receive and transmit packets without any VLAN information.

3.3 MAC Rewriting

The MAC Rewriting approach focuses on a closed data center environment. It allows the grouping of virtual machines into isolated virtual network segments and enforces policies to control network packet flow between and inside these groups. The underlying physical network is a completely switched network and assumed to be a "constrained world" hosted by a single owner who controls the infrastructure. However, certain virtual machines may be allowed to communicate with external systems through a NAT gateway.

3.3.1 Networking Design

In contrast to both VLAN Tagging and EtherIP encapsulation that virtualize the network at layer 2, the MAC Rewriting approach virtualizes the network based on layer 3 network-level information and provides the abstraction of **farms** which consist of several **subnets**. For example the IP address space is segmented by assigning IP addresses of the format $10.<FARM>.<SUBNET>.<HOST>$ to virtual machines. By default, VMs inside a subnet can communicate with each other without any restrictions. However, communication between VMs of different subnets has to be explicitly allowed by the farm owner. Communication across two different farms is only permitted if both sides have mutually agreed on such a communication. At the core of the farm network is a *notional gateway* connecting all of the subnets within the farm. The gateway mediates inter-subnet communications within the farm and inter-farm communication across multiple farms. However, the gateway does not exist; we only provide the illusion of it to virtual machines. In fact its actual functionality is distributed amongst all the physical machines hosting VMs within the farm. Figure 5 visualizes this architecture.

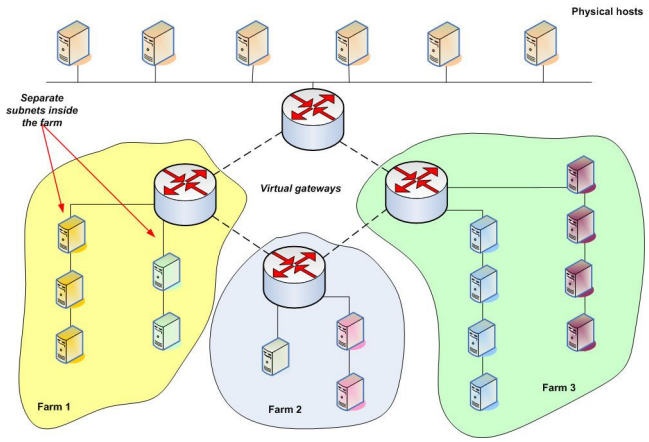


Figure 5: Farms, subnets and virtual gateways.

MAC rewriting. As in the virtual switch approach, each host OS of the virtualized infrastructure runs a component that intercepts packets emitted by virtual machines. This component takes care of eventually rewriting Ethernet addresses of packets. Addresses are rewritten to (1) provide the illusion of a gateway, (2) not allow VM MAC addresses on the physical wire, and (3) map broadcast and multicast packets into farm- or subnet-specific multicast traffic.

To provide the illusion of a gateway, if the sending and receiving VM are located on different subnets, then the gateway's MAC address is set as source MAC address inside the packet before it is passed into the destination VM. Therefore VMs never see MAC addresses of VMs that are hosted on a different subnet.

To prevent MAC addresses from appearing on the wire, if a packet has to go on the wire to a VM that is hosted on a remote system, then the virtual source Ethernet addresses in the packet header is replaced by the Ethernet address of the sending physical machine. Further, the virtual destination Ethernet address is replaced by the Ethernet address of the physical machine that is hosting the destination VM. On the destination host, the physical MAC addresses are substituted by the virtual MAC addresses before the packet is passed into the destination VM. This means that no virtual MAC addresses will ever be seen on the wire, and VMs never see MAC addresses of physical machines. Figure 6 illustrates how a packet is typically processed under this scheme.

Lastly, to do the mapping, when a VM emits broadcast or multicast packets, MAC addresses have to be rewritten to preserve isolation.

Address mapping. Each host OS manages a lookup table that maps a virtual IP address to a virtual MAC address. It also stores the physical MAC address of the machine hosting that particular virtual IP. For this purpose an ARP solicitation engine runs on each physical machine to

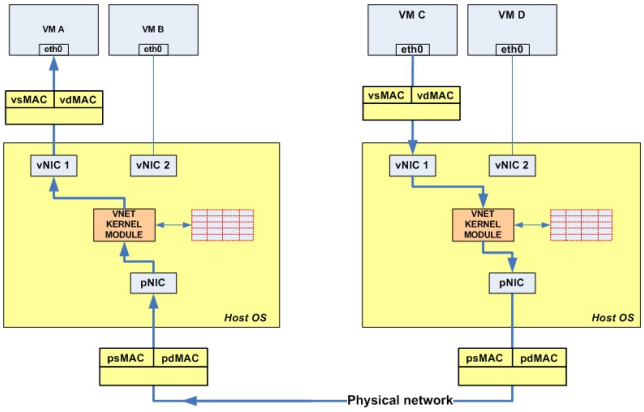


Figure 6: MAC Rewriting process.

quickly discover the location of virtual machine IP addresses. Additionally it takes care of keeping involved machines up-to-date when IP addresses move between VMs or when a VM is relocated to a another host within the infrastructure. When a packet arrives on a destination machine, it might happen that this host does not have a virtual MAC address binding for the packet's virtual source IP address, because virtual MAC addresses get lost through the MAC Rewriting processes when a packet has to go out on the physical wire. However, the destination host can quickly find out about the unknown binding by sending a *"backwards" unicast request* to the machine on which the sending VM is located. This machine is identified by the packet's source Ethernet address.

Broadcast / multicast mapping. *Broadcast* IP packets have to be delivered to all VMs in the same subnet, so they possibly have to be passed to multiple physical machines. We use Ethernet multicast addresses for that purpose - each subnet is represented by a multicast group, and physical machines that host VMs on that subnet have to register to that particular multicast address. Whenever a VM emits a broadcast packet, the host OS makes use of MAC Rewriting and replaces the broadcast destination MAC address with the subnet's multicast address before letting the packet on the wire. The same is applicable for farms, and packets that have to be delivered to a whole farm are sent to the multicast group of that particular farm. IP *multicast* traffic coming from virtual machines is either mapped into a subnet multicast address and delivered only subnet-wide, or handled as a farm-wide multicast distribution. This depends on whether the IP packet is allowed to pass subnet boundaries - which is determined from its TTL value.

Packet filtering. Network packet flow policies are managed by a packet filtering component that is distributed amongst all physical machines. Packet filtering is enforced within the host OS and therefore cannot be bypassed by any VM. We use packet filtering to control and restrict inter-subnet or inter-farm communication. Additionally, through this a farm owner can specify more sophisticated network policies on a per-VM basis.

3.3.2 Requirements Revisited

Layering over arbitrary physical networks. Currently we rely on having a flat layer 2 network as underlying physical infrastructure and do not support any other network topologies. This is mainly because the solicitation mechanism is based on ARP and therefore does not cross subnet boundaries.

Routing within virtual networks. Communication between different subnets is possible if this path has been permitted by the farm owner. There is no need for a routing VM as in the VLAN Tagging or EtherIP encapsulation approach, because we provide the illusion of that intermediate hop through MAC Rewriting. VMs of two different farms may also communicate with each other - if both farm owners have mutually agreed on such a communication. Through the MAC Rewriting process packets appear to a VM as if they have traversed a gateway, but in reality packets are passed between VMs with just a single network hop. Subnet and farm boundaries are enforced by a packet filtering engine that runs on each host and controls packet flow for virtual machines.

Gatewaying to non-virtualized systems. Virtual machines can communicate with non-virtualized systems that do not run our network virtualization technology (assuming that this communication is not restricted through packet filtering). The ARP engine takes care of discovering locations of any kind of machine within the infrastructure. Packets coming from these non-virtualized systems are processed using MAC Rewriting in the same way as explained before, there is no special treatment necessary.

4. COMPARATIVE STUDY

In this section, we assess the advantages and disadvantages of each approach in different usage scenarios. In Section 4.1, we discuss the networking features provided by each approach. In Section 4.2, we compare the approaches with respect to their ability to support network isolation. In Section 4.3, we present a number of scenarios that each technology can be best targeted for. We argue that, in many ways, our approaches are complementary and can be incorporated into one solution that provides a secure and efficient LAN / WAN virtual networking service. We report a summary of the advantages and disadvantages of the different approaches in Table 1.

4.1 Networking Features And Limitations

All three approaches enable network virtualization; to do so, however, they use different mechanisms that require different assumptions and dependencies.

4.1.1 Support From Physical Infrastructure

Our network virtualization approaches are mostly software-based; however, the VLAN tagging approach requires the use of hardware devices for network separation. In particular, VLAN tagging requires support from physical network switches to isolate virtual machine network traffic. Thus, hardware switches need to be able to understand the VLAN information and process tagged packets accordingly. The switches need to be (re-)configured with VLAN information whenever VMs are deployed, removed or migrated – which can happen frequently within dynamic and flexible infrastructures. It is thus a challenge to manage and alter the configuration of physical switches in an automated manner. Another issue with VLAN tagging is that modern NICs support VLAN offloading to accelarate network packet processing. As a result, they can potentially strip the tags from packets on reception. This feature needs to be disabled on

all hosts because it interferes with our technology as we explain in Section 5. In contrast, both EtherIP encapsulation and MAC Rewriting are purely software-based and do not require any specific configuration of physical network devices or NIC drivers.

Each approach can be deployed on a restricted set of network types. Network virtualization based on MAC Rewriting or EtherIP encapsulation relies on having an IP network deployed between physical machines of the virtualized infrastructure. The MAC Rewriting approach further needs to run on a flat layer 2 network within a LAN environment and VMs have to be given IP addresses of a predefined format. The EtherIP encapsulation approach in contrast can transmit packets over any arbitrary physical network topology that supports multicast. The VLAN tagging technology can be deployed within IP and non-IP networks, but is limited to a LAN infrastructure. Note that multiple approaches can be deployed on a data-center in a complimentary fashion, e.g., to enable fast LAN connectivity using VLAN tagging and a relatively slow WAN connectivity using EtherIP.

Lastly, the approaches differ in the way they handle virtual machine MAC addresses. In VLAN tagging, MAC addresses of virtual machines are allowed to appear on the physical network which increases the amount of entries within network switch tables when deploying large numbers of VMs. This problem is eliminated in networks that use the MAC Rewriting or the EtherIP encapsulation approach.

4.1.2 Networking Support for Virtual Machines

Each approach supports a restricted set of network protocols. In particular, MAC Rewriting only supports IP-based networking for VMs. Further, it currently requires that all VMs have IP addresses of a defined format, for example, we currently assign addresses of the form 10. $< FARM >$. $< SUBNET >$. $< HOST >$ to virtual machines. While we do require some addressing structure that does not allow completely arbitrary user-selectable IP addresses, we do not impose any restrictions on the prefixes or address ranges to use. For example, if a customer provides us with his or her own class B network, then we can use those addresses instead of a 10. $< FARM >$. $< SUBNET >$. $< HOST >$ scheme. EtherIP and VLAN tagging do not pose any restrictions on layer 3 networking protocols; they can be used when virtual machines want to run non-IP based services.

The particular advantage with the MAC Rewriting approach is that every communication path between two virtual machines only ever involves a single network hop. For example, if two communicating VMs are located on different subnets (or even on different farms), network traffic does not have to pass a router (or a routing VM) as in the EtherIP approach. This yields better network performance and additionally decreases the management effort for deploying and configuring routing entities. Similarly, in VLAN tagging we can avoid the deployment and management of routing VMs since the physical network switches are configured to perform fast inter-VLAN routing.

Lastly, MAC Rewriting does not require any form of packet encapsulation and therefore allows VMs to use full MTU sized packets that can potentially yield higher network throughput. EtherIP significantly reduces the MTU of virtual machine network packets: when encapsulating an Ethernet frame within an IP packet it reduces the MTU available to VMs by 38 bytes per packet. In an ideal setup, VLAN tagging does not reduce the MTU of packets at all, because the IEEE 802.1Q VLAN tagging standard allows the addition of 4 bytes per VLAN header to the normal MTU. In general, most of today's network switches support much higher MTUs, and so in many cases the problems stated here can eliminated[2].

4.1.3 Manageability

Unlike MAC rewriting, both EtherIP and VLAN tagging need external network entities to be managed accordingly. The EtherIP approach requires that routing VMs are set up within the infrastructure. These need to be deployed just like other virtual machines, but also need to run specific routing software which needs to be programmed and controlled. One drawback of this is that a routing VM can only interconnect as many VNETs as the virtualization software allows a VM to have virtual network interfaces, e.g. when using Xen this limit is 3 vNICs per guest and when using VMWare the limit is 4 (Server 1.0 and ESX version). The VLAN tagging approach does not necessarily require routing VMs, but instead relies on physical switches to be configured with VLAN support. Hence setting up a new virtual network always involves deploying new switch configurations within the physical infrastructure. To enable automated deployment of virtual networks this means that management software needs to be able to communicate with hardware switches (of possibly different manufacturers) and VNET components on host machines.

Configuring new virtual networks when using the MAC Rewriting approach does not involve configuration of network devices or setting up of routing VMs. However, IP addresses of virtual machines need to be managed and allocated in a global and secure manner to ensure that virtual network boundaries and packet flow restrictions can be enforced properly - which is only possible in a very restricted and highly controlled data center environment. However, binding VMs to use IP addresses of a specifically defined format significantly reduces this management effort.

4.1.4 Scalability

All three approaches pose scalability limitations due to their particular designs. In the MAC Rewriting approach, VMs are assigned to a farm/subnet combination through their IP addresses which have the form 10. $< FARM >$. $< SUBNET >$. $< HOST >$. Because the IP address is only a four-byte value, the number of farms and subnets that we can encode in an IP address is limited, and this defines the number of farms and subnets that we can host on a single network. The current design provides up to 14k subnets to customers. Additionally, it is possible to connect multiple physical networks together in a federated manner at the expense of additional hops in the network path which can have a significant impact on the performance. Extending the MAC Rewriting approach to an IPv6 solution will break the current limits of 32-bit IP addresses, and hence will provide a much larger number of farms and subnets within a single network.

We currently use the 802.1Q standard ([11]) when deploying our VLAN tagging solution. 802.1Q defines a four-byte header that is placed on Ethernet frames. Within this

[2]In addition to support from the physical network infrastructure, the network cards on all systems also need to be able to handle larger MTU sizes.

header there is a 12-bit field for the VLAN identifier. This means that a solution that is based on VLAN tagging can only provide up to 4096 VLANs (or VNETs) within a single local network. A possible option to overcome this limit would be to stack multiple VLAN tags in order to expand the VLAN space - this is supported by some hardware devices (mainly Cisco) through the IEEE 802.1q-in-q (QinQ or stackable VLANs) technology.

The EtherIP approach poses a similar scalability limitation in terms of the number of VLANs that can be supported. EtherIP uses the standard in [12] that adds a 16-bit header to an Ethernet or 802.3 frame. This header includes a four-bit field that specifies the EtherIP version number, and leaves 12 bits reserved which we use to encode the VNET identifier. This implies that the EtherIP network virtualization approach is limited to 4096 VLANs within a single administrative domain. Another scalability issue of this approach is the use of routing VMs. Routing VMs interconnect virtual networks by having multiple network interfaces (one per VLAN) and routing traffic between those. The number of network interfaces that a routing VMs can have is limited by the virtualization technology in use (e.g., Xen or VMware). Additionally, a routing VM represents a single point of failure and can potentially become a bottleneck for inter-VNET communication because all traffic has to traverse that VM. Therefore, in a large-scale virtualized infrastructure it is critical that traffic is efficiently load-balanced across various routing VMs.

4.2 Security Analysis

Machine virtualization alone provides immediate isolation of computing resources such as memory and CPU between guest domains. However, the network remains to be a shared resource as all traffic from guests eventually pass through a shared network resource (e.g., a virtual switch) and end up on the shared physical medium. As a result, we need mechanisms to (1) control the information flow between virtual machines (e.g., who can communicate with whom), (2) configure virtual and physical network resources, and (3) separate network resources used by each networking domain. In this section, we revisit the security requirements outlined in Section 2 and compare each approach in terms of their capability to enforce and support domain policies. In particular, we describe how each approach reinforces network separation and isolation within and across network domains.

Our solutions mostly rely on VLAN separation and address mapping techniques to control the information flow within and across networking domains. The approaches differ in ways the address mapping and packet forwarding decisions are taken. In particular, EtherIP and VLAN tagging use a single virtual switch per VLAN segment that forwards and accepts traffic to and from member hosts. MAC rewriting implements a distributed virtual gateway to (re)direct traffic within and between networking domains.

Within a networking domain (i.e., a VLAN segment or a subnet), EtherIP uses a single (distributed) virtual switch to decide on which network packet to forward and accept. Similarly, MAC rewriting uses a single (distributed) virtual gateway to do same. In contrast, in VLAN tagging a physical switch is additionally involved in the decision process. Thus, the former two have an advantage over the latter because the decision is taken on an end-to-end basis and no intermediate node (e.g., the physical switch) is involved. Involving the physical switch, however, can provide better network separation as compared to software-only approaches. As a result, VLAN tagging can provide network isolation not only between virtual machines, but also between physical machines.

Another basis for comparison is the way each approach handles the communication across different networking domains. In MAC rewriting, all decisions on whether to allow communication across domains (i.e., subnets and farms) are taken by a distributed virtual gateway. This yields an efficient scheme and allows VMs to communicate over a single hop regardless of their domain membership. However, the virtual gateway becomes the single point of decision that needs to be trusted by all domains. In contrast, in EtherIP and VLAN tagging, a virtual switch is designated to a particular domain and controls communication within that domain only. Communication across domains is handled by gateway routers or firewalls that are trusted by both domains. This yields a less efficient scheme than MAC rewriting. However, it can potentially provide a better separation of networking resources across domains. Further, the virtual switch design allows the disaggregation of components into separate, highly isolated domains. For example, each virtual switch can run in a separate purpose-build strippeddown VM. This creates an advantage for providing high assurance about the security properties of a system and enables the separation of the configuration and management of VLANs that are not under a single administrative entity. This is especially beneficial for systems that are shared amongst competing parties.

While the vSwitch approaches focus on strong separation of virtual networks, the MAC Rewriting technology addresses secure inter-farm communication from a different perspective. In particular, this approach fits best when customers owning different farms want to share services in a controlled but efficient manner. This is, for example, the case with the Service Utility Platform (SUP) project which is currently developed within HP Labs. We present a MAC Rewriting prototype that runs on SUP in Section 5.

Lastly, all schemes support confidentiality and can provide encryption over untrusted physical medium. Further, EtherIP can protect the VLAN membership information using end-to-end encryption. In VLAN tagging, however, the tag needs to be revealed to the physical switch. Hence, end-to-end encryption of the tagging information is not possible. MAC rewriting does not involve a tag, hence no extra protection is needed.

4.3 Applications and Use Cases

Each networking technology we have described in the previous sections has specific advantages in different environments with varying use cases and scenarios. Here, we present a number of scenarios that each technology can be best targeted for.

Customers might want to use virtual machine technology to run legacy applications, simulate proprietary network solutions, or run non-IP based protocols. The MAC Rewriting approach cannot be used for these specific application requirements, because it only supports IP-based network communication. Note that most mainstream data center solutions will fit for the MAC Rewriting approach in this regard (as IP is a widely used protocol). However, it might not be the best applicable solution for customers that

Table 1: Summary of the advantages and disadvantages.

Features	MAC RW	EtherIP	VLAN Tagging
Arbitrary L3 protocols for VMs	N	Y	Y
Forces reduced MTU sizes	N	Y	Y/N[a]
User-selectable IP addresses for VMs	N	Y	Y
Single-hop networking	Y	N	N
Arbitrary L3 protocols for hosts	N	N	Y
Arbitrary physical network topology	N	Y	Y
Requires programming of physical switches	N	N	Y
Requires support from NICs	N	N	Y
Runs over WAN link if tunneling available	Y	Y	Y
VM MAC addresses on the wire	N	N	Y
Requires external routing entities	N	Y	Y
Automatic network policy verification[b]	Y	N	N
Integrated NAC framework[c]	N	Y	Y
Number of subnets per single LAN (by default)	14K	4K	4K

[a]As defined in the IEEE 802.1Q standard, a VLAN-tagged Ethernet frame can be larger than the standard 1500 byte Ethernet MTU size. However, many older network interfaces cannot handle frames with sizes of over 1500 bytes, so in practice the standard MTU size has to be reduced for most older network cards.
[b]As described in 3.3
[c]The NAC framework is a Network Admission Control mechanism that we have implemented in order to control whether or not a VM is allowed to join a specific virtual network segment.

require customized network configurations (e.g., non-IP).

Currently, the VLAN approach is the best fit when running over a shared, untrusted physical network infrastructure as it facilitates the separation of network traffic on the physical wire – enforced by network switches within the infrastructure. However, both EtherIP and MAC Rewriting can be easily enhanced to securely run over untrusted networks by using encryption or network access control mechanisms such as, for example, IEEE 802.1X ([1]). As well as being able to use 802.1X network access control to restrict physical machine access to a particular physical LAN, the virtual switch implementation supports 802.1X authentication of virtual machines before they are allowed to join a particular virtual LAN.

Lastly, the MAC Rewriting prototype implementation facilitates a scheme to automatically verify network policies and provide assurance about the actual state of the complete system from a network point of view. This is an important feature that can help guarantee fulfilling certain security requirements (e.g., as defined in Service Level Agreements (SLAs)).

5. PROTOTYPE IMPLEMENTATIONS

In this section, we provide a brief overview of two prototypes we implemented on Xen and VMware platforms using the virtual networking technologies we introduced in Section 3. In Section 5.1, we present the virtual switch implementation that uses a combination of EtherIP encapsulation and VLAN tagging. In Section 5.2, we present the virtual gateway implementation that uses MAC Rewriting. Lastly, in Section 5.3, we empirically assess the performance of each approach.

5.1 Security-enhanced Network Virtualization

We describe a Xen-based [3] prototype implementation of the virtual switching framework. Figure 7 shows the implementation of virtual switches that manage two vLAN segments, $VLAN_\alpha$ and $VLAN_\beta$. The policy engine, also shown in the figure, implements the policies corresponding to the security domains formed by the VLAN segments.

5.1.1 Implementation Details

Our implementation is based on Xen-unstable 3.0.4, a VMM for the IA32 platform, with the VMs running the Linux 2.6.16 operating system. Our networking extensions are implemented as kernel modules in Dom0, which also acts as driver domain for the physical NIC(s) of each physical host. A driver domain is special in the sense that it has access to portions of the host's physical hardware, such as a physical NIC.

The virtual network interface organization of Xen splits a NIC driver into two parts: a front-end driver and a back-end driver. A front-end driver is a special NIC driver that resides within the kernel of the guest OS. It is responsible for allocating a network device within the guest kernel (eth0 in Dom1 and Dom2 of hosts A and B, shown in Figure 7). The guest kernel layers its IP stack on top of that device as if it had a real Ethernet device driver to talk to. The back-end portion of the network driver resides within the kernel of a separate driver domain (Dom0 in our implementation) and creates a network device within the driver domain for every front-end device in a guest domain that gets created. Figure 7 shows two of these back-end devices, vif1.0 and vif2.0, in each of the two hosts A and B. These back-end devices correspond to the eth0 devices in Dom1 and Dom2, respectively, in each host.

Conceptually, the pair of front-end and back-end devices behaves as follows. Packets sent out by the network stack running on top of the front-end network device in the guest

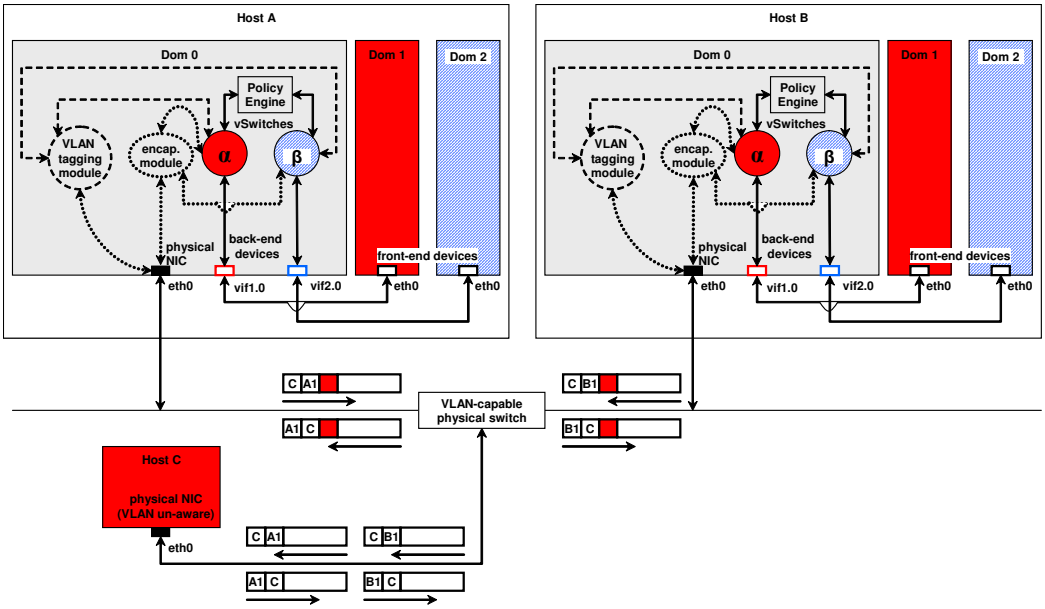


Figure 7: Prototype implementation of VLANs as security domains.

domain appear as packets received by the back-end network device in the driver domain. Similarly, packets sent out by the back-end network-device by the driver domain appear to the network stack running within a guest domain as packets received by the front-end network device. In its standard configuration, Xen is configured to simply bridge the driver domain back-end devices onto the real physical NIC. By this mechanism, packets generated by a guest domain find their way onto the physical network and packets on the physical network can be received by the guest domain.

The Xen configuration file is used to specify the particular vSwitch and the particular port in the vSwitch to which a Xen back-end device is attached. We use additional scripts to specify whether a particular vSwitch should use one or both of VLAN tagging and encapsulation mechanisms for isolating separate virtual networks.

The vSwitches for $VLAN_\alpha$ and $VLAN_\beta$ are each implemented in a distributed fashion (i.e., spread across hosts A and B) by a kernel module in Dom0, which maintains a table mapping virtual network devices to ports on a particular vSwitch. Essentially, the kernel module implements EtherIP processing for packets coming out of and destined for the VMs. Each virtual switch (and hence VLAN segment) has a numeric identifier associated with it. The Ethernet packets sent by a VM are captured by the kernel module implementing part of the vSwitch as they are received on the corresponding back-end device in Dom0. The packets are encapsulated using EtherIP with the network identifier field set to match the identifier of the vSwitch that the VM is supposed to be plugged into. The EtherIP packet is given either a multicast or unicast IP address and simply fed into the Dom0 IP stack for routing onto the physical network. The kernel module also receives EtherIP packets destined for the physical host. The module un-encapsulates the Ethernet frames contained in the encapsulated EtherIP packets and transmits the raw frame over the appropriate virtual network interface so that it is received by the intended guest vNIC.

In addition to the kernel module for EtherIP processing, we have also implemented a kernel module for VLAN tagging in Dom0 of each virtualized host. Ethernet packets sent by a VM are grabbed at the same point in the Dom0 network stack as in the case of EtherIP processing. However, instead of wrapping the Ethernet packets in an IP packet, the VLAN tagging module tags the packet with the ID of the VLAN that the VM is supposed to be connected to. The tagged packet is then sent straight out onto the wire through the physical NIC. The VLAN tagging module also intercepts VLAN packets arriving on the physical wire destined for a VM. The module then removes the VLAN tags and, based on the tag, maps packets to the appropriate vSwitch (α or β) which, in turn, maps them to the corresponding back-end device (vif1.0 or vif2.0) in Dom0. The packets eventually arrive at the corresponding front-end device (eth0 in Dom1 or Dom2) as plain Ethernet packets.

5.1.2 *Implementation Issues*

Below are some implementation issues we had to tackle in realizing the VLAN and encapsulation approaches.

1. Some Ethernet cards offer VLAN tag filtering and tag removal/offload capabilities. Such capabilities are useful when running just a single kernel on a physical platform, in which case there is no need to maintain the tags for making propagation decisions. However, for our virtual networking extensions, the hardware device should not strip the tags from packets on reception over the physical wire; instead, the kernel modules we have implemented should decide to which VM the packets should be forwarded. For this purpose, we modified the Linux kernel tg3.ko and forcedeth.ko network drivers so as to disable VLAN offloading.

2. For efficiency reasons, the Xen front-end and back-end driver implementations avoid computing check-

sums between them for TCP/IP and UDP/IP packets. We modified the Xen code to also handle our EtherIP-encapsulated IP packets in a similar manner.

3. The EtherIP encapsulation approach relies on mapping a virtual Ethernet broadcast domain to a IP multicast domain. While this works in a LAN environment, we encountered problems when creating VLAN segments that span WAN-separated physical machines. We resolved this issue by building uni-directional multicast tunnels between successive LAN segments.

5.2 Network Virtualization in Datacenters

5.2.1 Providing Secure Service Environments

A prototype of the MAC Rewriting approach has been implemented as part of the Service Utility Platform (SUP) project within HP Labs. The SUP is a secure, automated management system that facilitates the unified aggregation of virtual resources into secure service environments. The SUP focuses on a local virtualized datacenter. In this context the MAC Rewriting approach securely separates and isolates customers' virtualized elements to provide the illusion of dedicated network resources. It also separates customer networks from the utility network. Additionally, customer farms should be allowed to implement and selectively expose services to other farms and consume services provided by other farms, to allow a rich ecosystem of interacting services to develop. The MAC Rewriting approach is the most suitable choice to enable network virtualization in this particular scenario, because unlike the other two approaches, the communication that spans over farm boundaries is realized as a single-hop network path.

5.2.2 Security Assumptions

In implementing MAC rewriting, we have assumed that the following conditions are present:

1. The datacenter network is separated from the Internet by a firewall and NAT, only a subset of VMs will be visible externally.

2. A layer 2 connection exists between all machines that are part of the virtualized datacenter. However, tunneling could allow federation of multiple instances of these datacenters.

3. The virtualized datacenter is controlled by a single owner who manages physical machines and network devices of the infrastructure. All hosts run standard software to provide basic system-level virtualization. We use Xen and VMWare for this purpose.

4. Access to the host OS on VMWare systems as well as to Xen's most privileged domain is assumed to be granted only to authorized users, as in here one has full control over the configuration of virtual networking capabilities.

5.2.3 Implementation

Every physical machine of the infrastructure runs a virtualization layer - in our cases Xen 3 or VMWare Server. The virtual networking capabilities are provided by a Linux kernel module (the VNET module). On a Xen host, this module runs in domain 0 (or in an isolated driver domain) and on a VMWare system it is loaded within the host OS. Here each VM's network card is represented by a virtual NIC, and also the driver for the physical NIC resides here. In the current design all network traffic has to pass through this domain, and it is impossible for VMs to circumvent this.

The VNET module (1) intercepts packets coming from virtual machines as well as coming from the physical wire, (2) hands them to the filtering engine, (3) replaces MAC addresses, and (4) passes them on to either one or more VMs or onto the network to a remote machine. The kernel module hooks into the network stack at the Ethernet layer for packets from / to the physical network and from / to VMs. Packets from the host OS (or Xen domain 0 respectively) are intercepted on the IP layer and also passed back in at that level. This means that the host OS does not do any ARP discovery on its own, but uses the shared ARP engine provided by the VNETLinux kernel module instead.

The kernel module exports certain configuration parameters and options to user-space tools through the Linux SYSFS filesystem. Only authorized users or processes can access this interface to configure virtual network settings. SYSFS cannot be directly accessed from remote machines, therefore there is a control daemon running on each system to enable automatic remote deployment.

The current implementation supports all features of the MAC Rewriting approach that have been described in Section 3. Although, we do not provide transparent encryption of network traffic yet.

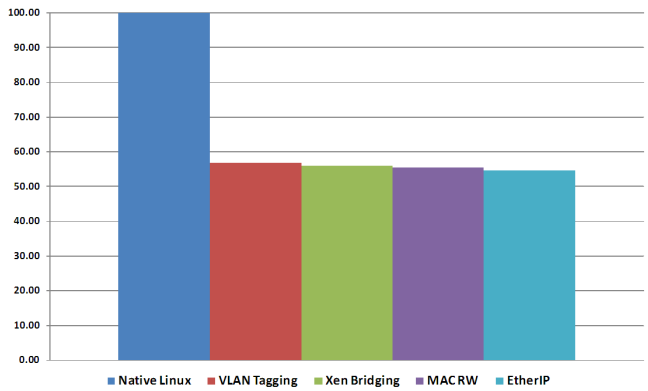
Packet filtering is currently realized using standard IPTables and a proprietary matching module that allows us to easily filter traffic to enforce farm and subnet boundaries. Network policies have to be set up within the host OS using standard IPTables user-space tools. It is also possible to configure more advanced network policies on a per-VM basis - here we support most filtering options that IPTables provides. Policies can only be set up by the owner of the VM.

Global packet filtering rules that enforce farm and subnet boundaries are distributed amongst all participating host machines. Each host runs a rules management component that knows about illegal packet flows and immutable, global policies (these static rules are part of the configured Trusted Computing Base (TCB) that all physical machines boot up with). All new network policies have to be validated and approved by this component. Additionally, policies that affect multiple farms with potentially different owners always have to be mutually agreed.

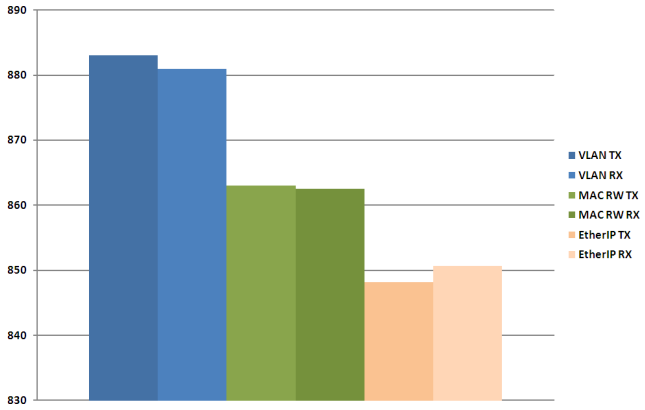
5.3 Performance Analysis

As part of our comparative study, we have empirically assessed how current prototypes fare when running basic networking applications within a virtual machine. We compare our solutions against a virtualized system (such as Xen or VMWare) in standard bridging mode and also against a non-virtualized system. We first report and analyze the performance figures for each VNET technology and compare the results. We then provide an overview of potential performance improvements that we are currently investigating.

We obtained the throughput results using the `netperf` network benchmark and latency results using the `ping` tool.



(a) Summary of relative throughput performance results.



(b) TX/RX throughput benchmark results in Mbps.

Figure 8: Intra-subnet Throughput Results.

While our VNET technologies run with both VMWare and Xen systems, we only provide performance results for a Xen setup in this report as both technologies perform very similar. Our test systems run Xen 3.0.4 (release version) with a 2.6.16.33 Linux kernel in domain 0. We used HP ProLiant BL25p G2 blade servers each fitted with two AMD Opteron processors running at 2 GHz, 8GB system memory and a Gigabit Ethernet card. For throughput measurements we have configured netperf with a confidence level of 99% and a confidence interval of 5% to ensure that our results only include consistent samples. As netperf is a client/server-based tool, we ran one instance of the benchmark on one guest VM as a server process and another instance on the second guest VM to do the actual benchmark.

We have measured network performance for two different scenarios:

1. Intra-subnet communication - both communicating VMs reside on the same virtual network segment (or subnet), and
2. Inter-subnet communication - communicating VMs reside on different subnets.

5.3.1 *Intra-subnet communication*

Figure 8(a) shows a summary of the throughput results that we have obtained for a netperf *TCP_STREAM* test using a message size of 8192 bytes and a socket size of 65536 bytes. The figure evaluates throughput performance of a standard Xen bridged configuration, VLAN tagging, EtherIP encapsulation and MAC Rewriting and compares them to the performance that a native (non-virtualized) Linux system achieves. These results take into account how much CPU resources each approach consumes during the tests. This is a critical characteristic of any solution that needs to be assessed, mainly because it is important to clearly account for resource utilization in a virtualized system where resources like CPU are shared across multiple VMs. The results show that in this setup all approaches perform very similarly.

In Figure 8(b), we report more detailed throughput measurements that break down the implementation specific differences between our three network virtualization approaches. These results show the actual achieved throughput reported by the netperf tool in Megabits per second (Mbps). We do not report CPU figures in these graphs as utilization between these three technologies is very close and therefore does not give any additional insight. The graphs show that the VLAN tagging extension achieves the best performance overall while EtherIP encapsulation yields the worst throughput.

All VNET extensions perform better on the **Tx path** - except the EtherIP method where the major cost is having to allocate a fresh socket buffer (`skb`) and copy the original buffer data into the fresh `skb`. During the initial allocation of the `skb`, the Linux network stack allocates a fixed amount of headroom for the expected headers that will be added to the packet as it goes down the stack. However, not enough space is allocated initially to accommodate the EtherIP header; thus, we have to copy the data, which is a costly operation. However, there is *some* spare headroom space, which is sufficient for the extra VLAN tag. As a result, the VLAN tagging method does not suffer from the packet copying overhead. It is important to mention here that the small headroom is an implementation specific issue that could be fixed with a specifically patched Linux kernel. The MAC Rewriting approach does not change the packet in any way other than eventually replacing source and destination Ethernet addresses. Therefore, it does not need to allocate a new socket buffer. However, it has more complex decision paths in order to find out how to rewrite the packet which results in more processing overhead. Additionally, packets need to be processed up to the IP header while both VLAN tagging and EtherIP encapsulation inspect packets on Ethernet level only.

In the **Rx path**, there is no packet-copying overhead for the EtherIP approach; the extra EtherIP header merely has to be removed before the packet is sent to a VM. However, as compared to VLAN tagging and MAC Rewriting in which packets are grabbed from the Linux network stack immediately after coming into the OS from the network driver, the EtherIP solution requires that packets are passed into and processed by the host OS IP stack before they are handed over to the EtherIP packet handler of the virtual switch

code. Lastly, the MAC Rewriting approach performs similar for both Tx and Rx due to similar packet processing - e.g. the same lookups and the same packet rewriting in both directions.

An important difference between MAC Rewriting and the other approaches including standard Xen bridging is that MAC Rewriting virtualizes the network on a higher level (network layer). This has two significant implications from a performance point-of-view:

1. MAC Rewriting manages larger lookup tables. In fact the VLAN / encapsulation implementation facilitates a table per vSwitch/VNET while the MAC Rewriting prototype manages only one table per physical host (as it operates in a distributed router fashion).

2. Communication between VMs which are deployed on an infrastructure that runs MAC Rewriting only ever involves a single network hop. All other approaches require a routing entity in the network path in the case where communicating VMs are on different VNETs.

Lastly, in Table 2 we report the round-trip times between two guest VMs on a physical host for the bridged, VLAN, EtherIP encapsulation, and MAC Rewriting cases obtained using the `ping -c 1000 host` command, i.e., 1000 packets sent. The results show that on average Xen bridging has the lowest round-trip time.

Table 2: Intra-subnet Round-trip Times using Ping.

	Minimum	Average	Maximum	Mean Deviation
Bridged	0.1355	0.18	0.294	0.0235
MAC RW	0.1582	0.2062	0.3182	0.0276
VLAN	0.1395	0.21225	0.35675	0.0295
EtherIP	0.151	0.246	0.378	0.0335

5.3.2 *Inter-subnet communication*

We have additionally measured and analyzed the performance that our different approaches achieve when the communicating VMs reside on different virtual network segments (or subnets). While all technologies offer similar performance for communication within a subnet, the results for inter-subnet communication demonstrate a major design difference between MAC Rewriting and the other two extensions, and its impact on network performance. This is because MAC Rewriting operates at L3 and it does not rely on external routing entities, whereas both VLAN tagging and EtherIP encapsulation introduce a routing entity on the network path for inter-subnet communication.

We cam deploy routing entities in various forms for both the EtherIP encapsulation and the VLAN tagging approach. These can mainly be distinguished by the degree of flexibility, manageability and performance that they can provide. For VLAN tagging we enforce routing between virtual network segments through the integration of layer 3 switches that are capable of carrying out Inter-VLAN routing. This approach allows high-performance inter-subnet communication. For EtherIP we can deploy Routing VMs that route packets between VMs that reside on different subnets. However, the use of Routing VMs introduces a huge performance

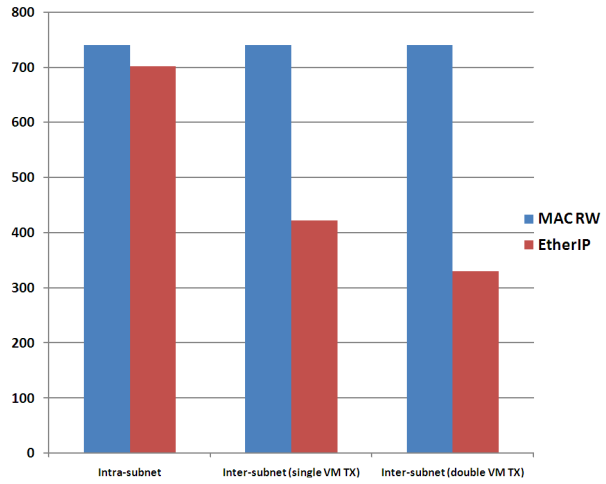


Figure 9: Inter-subnet Throughput Results in Mbps.

drop for cross-subnet traffic. This is demonstrated in Figure 9 which compares cross-subnet throughput performance of MAC Rewriting and EtherIP encapsulation. These results have been recorded with the same netperf test configuration and process that we have used for intra-subnet communication as described in Section 5.3.1. While both approaches achieve similar performance for network traffic that stays within a single subnet, we see a drop in performance of about **40%** in case of EtherIP encapsulation when traffic has to go across subnet boundaries and only a single VM is transmitting through the Routing VM at the same time (shown as "single VM TX" in Figure 9). The performance suffers even more when multiple VMs communicate through the a Routing VM at the same time: we have recorded a drop in throughput of almost **60%** when only two VMs are transmitting at the same time (shown as "double VM TX" in Figure 9). In contrast, the L3 approach of MAC Rewriting eliminates the need for any kind of external routing entities, and therefore it provides more constant and reliable network performance within and across subnets.

The introduction of Routing VMs on the network path also shows a significant increase in latency which we report in Table 3. The results show that latency is almost doubled when using a Routing VM for the EtherIP encapsulation extension, whereas no increased latency is reported when using MAC Rewriting.

Table 3: Inter-subnet Round-Trip Times using Ping.

	Min	Avg	Max	Mean Dev
MAC RW	0.1582	0.2062	0.3182	0.0276
EtherIP	0.287	0.37	0.4975	0.387

The poor performance of Routing VM network communication is mainly due to the fact that the overhead of virtualization hits at least twice. Also, it is clear that Routing VMs can quickly become the bottleneck of inter-subnet networking, and additionally a single point of failure. To overcome some of these issues we can also deploy routing entities di-

rectly on physical host machines.

5.3.3 Future Performance Enhancements

Network virtualization provides a flexible scheme to separate customer networks in virtualized platforms; usually at the expense of performance. A recent development we are investigating in this area is the enhancement of hardware with better support for virtualization. This is especially relevant to networking because the virtualization layer is involved in all network I/O processing which significantly degrades performance as we have reported in Figure 8(a). Advanced hardware virtualization support can remove the privileged domain or host OS from the main data I/O path which will enable direct access to the network hardware for a virtual machine.

We are also investigating network resource control mechanisms that can allow better performance isolation across virtual machines. Briefly, this scheme enables throttling network bandwidth for VMs which in turn can improve the overall network performance within a virtualized infrastructure. Further, it protects the infrastructure in the presence of malicious/buggy VMs that could use up network resources and slow down other VMs or network services.

6. RELATED WORK

Previous work on virtualizing physical networks can be roughly grouped into two categories: those based on Ethernet virtualization (Layer 2) and those based on TCP/IP-level virtualization (Layer 3). Virtualization technologies such as Xen and VMWare also provide basic virtual networking in these two categories – these are referred to as `bridged` and `routed` configuration modes. However, these basic approaches alone do not provide sufficient isolation of network traffic between virtual network segments. Further, they lack the mechanisms that can enforce more sophisticated intra-VNET and inter-VNET network policies.

Although both categories include a substantial amount of work (e.g., [12, 2, 4, 8, 9, 16, 19, 20, 21]), few studies have an explicit security focus:

Ethernet Virtualization: Ethernet virtualization aims at transporting multiple Ethernet connections over a single physical medium. There are a large number of Ethernet tunneling protocols [9]. Local transport over a "trusted" wire is usually multiplexed using the well-established VLAN standard IEEE 802.1Q-2003. It adds virtual LAN tags to each Ethernet segment and enables separation of multiple networks. An example for high-performance Infiniband VLANs is given in [10]. In wide-area networks, VLAN tags are often not preserved. To overcome these restrictions, Ethernet encapsulation has been proposed as an alternative [12, 19, 8, 9]. Ethernet packets (including tags) are wrapped into TCP/IP packets. This enables the embedding of a virtual Ethernet network into a wide-area network. Unfortunately, the performance and scalability of the resulting system are limited.

TCP/IP-level Virtualization: TCP/IP-based virtualization is for example used for *Overlay Networks* that provide application-level network virtualization among participating hosts. An overlay network typically consists of hosts (physical or virtual), routers, and tunnels that serve as virtual links between the hosts. Several overlay designs have been introduced in the literature: PlanetNet VNET [16, 4], X-Bone [20], Resilient Overlay Networks [2], and the JXTA project [21]. The designs share the common goal of creating a virtualized network layer with a customized topology mapped onto the actual physical infrastructure. They differ in the underlying technology that enables the mapping, management of the technology, and the terminology used.

Overlay networks are most useful for implementing a virtual network topology on top of the physical topology. However, they are not suitable for systems with strong separation, isolation, and flow control requirements. As an example, although the PlanetLab VNET provides separation of network packets originating from different *slices*, the separation is merely enforced using the OS network services [4]. Similarly in JXTA, *peergroups* are used to group network peers and enforce certain isolation properties [21]. However, it is the network administrator's responsibility to enforce flow control policies across group boundaries as JXTA does not impose any specific flow control schemes for the sake of flexibility. Other shortcomings of overlay networks are complex management models, binary intra-group flow policies, and lack of inter-group flow control policies.

The VIOLIN project addresses a number of these deficiencies and enhances the traditional TCP/IP overlay networks to create mutually isolated distributed environments [13, 17]. The main idea is to provide each subsystem with a virtual IP world having its own address space. In particular, a VIOLIN is created on top of an overlay network (such as PlanetLab [4]) and consists of virtual hosts, switches, and routers. Communication between these entities is enabled through a User-Mode Linux (UML) implementation enhanced with UDP-tunneling for inter-host communication[3]. The VIOLIN model provides isolation between different VIOLINs, which in turn enhances mobility through location-independent addressing. Further, the model enables the customization of each VIOLIN with the desired technology (e.g., IPv6) without requiring a global deployment. A major disadvantage of VIOLIN is that the model completely disallows inter-VIOLIN communication rather than adopting a policy-based flow control scheme. In practice, it may be desirable for VIOLINs belonging to different organizations to interact with each other under certain flow control policies enforced at each VIOLIN boundary.

Previous solutions also offered network virtualization schemes that do not rely on overlay networking. *Spawning networks*, employ nested programmable networks to form a hierarchy of virtual networks that are isolated from each other [6, 7, 14]. The main idea is to enable parent networks to *spawn* child networks that utilize the parents' resources. The child networks then may or may not choose to inherit certain characteristics from their parents. The advantages are that the child networks can employ a specialized networking technology (e.g., a mobile-IP network) while inheriting basic network functionality from their parent. Further, they can spawn child networks of their own, forming a forest of networks.

Spawning networks utilize the Genesis network kernel [14] that enables the life-cycle management of each spawned network including the spawning capability. The Genesis kernel is a complex virtual networking kernel that needs to be installed on every physical domain that will potentially host spawning networks. The major downside is that this requires major changes to the existing network infrastructure.

[3]A Xen-based solution has recently been introduced [18].

7. CONCLUSIONS

In this report we have presented and analysed three HPL VNET technologies that we have successfully developed and deployed in different application scenarios. We have assessed the advantages and disadvantages of each approach and discussed that the choice of which network virtualization technology should be based on application and security requirements, and also on how much control we have over the underlying infrastructure and what kind of features it provides. We have shown that our approaches show measurable differences from a performance point of view. Further, they impose different restrictions on network applications and rely on different external capabilities in order to operate.

In the light of our findings, we deduce that an optimal solution would interoperate multiple network virtualization technologies in different settings. E.g., one could use MAC Rewriting or VLAN tagging within a LAN / data center environment, but use EtherIP encapsulation in cases the packets need to traverse a WAN link. Ideally, we want to be able to automatically deploy the underlying network virtualization approach based on a high-level specification that describes networking and application needs.

We are currently investigating emerging networking and virtualization technologies that we will employ to further enhance our approaches regarding performance, manageability and security. These enhancements include hardware-assisted virtualization and networking, advanced network resource controlling, network access control mechanisms, and automated configuration management of network equipment.

Acknowledgements

We thank our colleagues at IBM Zurich for collaboration in secure network virtualization within the OpenTC project [15]. More information on this work can be found in [5].

8. REFERENCES

[1] 802.1x: IEEE standard for local and metropolitan networks — port-based network access control. IEEE Standards, 2004. Revision of 802.1X-2001.

[2] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. 18th ACM Symposium on Operating Systems Principles (SOSP-2001)*, pages 131–145, New York, NY, USA, 2001. ACM Press.

[3] P. T. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. 19th ACM Symposium on Operating Systems Principles (SOSP-2003)*, pages 164–177, October 2003.

[4] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, L. Peterson, T. Roscoe, and M. Wawrzoniak. Operating Systems Support for Planetary-Scale Network Services. In *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI'04), San Francisco, CA, USA*, 2004.

[5] Serdar Cabuk, Chris Dalton, HariGovind V. Ramasamy, and Matthias Schunter. Towards automated provisioning of secure virtualized networks. In *Proc. 14th ACM Conference on Computer and Communications Security (CCS-2007)*, October 2007. To appear.

[6] A. T. Campbell, M. E. Kounavis, D. A. Villela, J. B. Vincente, H. G. De Meet, K. Miki, and K. S. Kalaichelvan. Spawning Networks. *IEEE Network*, 13(4):16–29, July-August 1999.

[7] A. T. Campbell, J. Vicente, and D. A. Villela. Managing Spawned Virtual Networks. In *Proc. 1st International Working Conference on Active Networks (IWAN '99)*, volume 1653 of *LNCS*, pages 249–261. Springer-Verlag, 1999.

[8] C. I. Dalton. Xen Virtualization and Security. Technical report, HP Security Office Report, August 2005.

[9] R. Davoli. VDE: Virtual Distributed Ethernet. In *Proc. 1st International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom 2005)*, pages 213–220. IEEE Press, February 2005.

[10] S. W. Hunter, N. C. Strole, D. W. Cosby, and D. M. Green. BladeCenter Networking. *IBM Journal of Research and Development*, 49(6), 2005.

[11] IEEE. Virtual Bridged Local Area Networks. Technical Report ISBN 0-7381-3662-X.

[12] IETF. EtherIP: Tunneling Ethernet Frames in IP Datagrams. RFC 3378.

[13] X. Jiang and D. Xu. VIOLIN: Virtual Internetworking on OverLay INfrastructure. In *Parallel and Distributed Processing and Applications*, volume 3358 of *LNCS*, pages 937–946. Springer-Verlag, Berlin, 2004.

[14] M. E. Kounavis, A. T. Campbell, S. Chou, F. Modoux, J. Vicente, and H. Zhuang. The Genesis Kernel: A Programming System for Spawning Network Architectures. *IEEE Journal on Selected Areas in Communications*, 19(3):511–526, March 2001.

[15] The Open Trusted Computing Project, 2006. Available from `http://www.opentc.net/`.

[16] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[17] P. Ruth, X. Jiang, D. Xu, and S. Goasguen. Virtual Distributed Environments in a Shared Infrastructure. *IEEE Computer*, 38(5):63–69, May 2005.

[18] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen. Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure. In *Proc. IEEE International Conference on Autonomic Computing (ICAC-2006)*, June 2006.

[19] A. Sundararaj and P. Dinda. Towards Virtual Networks for Virtual Machine Grid Computing. In *Proc. 3rd USENIX Conference on Virtual Machine Technology (VM 04), pp. 177-190*, 2004.

[20] J. Touch. Dynamic Internet Overlay Deployment and Management using the X-bone. *Computer Networks*, 36(2-3):117–135, 2001.

[21] B. Traversat, A. Arora, M. Abdelaziz, M. Doigou, C. Haywood, J-C. Hugly, E. Pouyoul, and B. Yaeger. Project JXTA 2.0 Super-Peer Virtual Network, 2003. `http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf`.