On the Design of Scalable, Self-Configuring Virtual Networks

David Isaac Wolinsky, Yonggang Liu, Pierre St. Juste,
Girish Venkatasubramanian, Renato Figueiredo
University of Florida
{davidiw, yonggang, ptony82, girish, renato}@acis.ufl.edu

ABSTRACT

Virtual networks (VNs) provide methods that simplify resource management, deal with connectivity constraints, and support legacy applications in distributed systems, by enabling global addressability of VN-connected machines through either a common layer 2 Ethernet or a NAT-free layer 3 IP network. This paper presents a novel VN design that supports dynamic, seamless addition of new resources with emphasis on scalability in a unified private IP address space. Key features of this system are: (1) Scalable connectivity via a P2P overlay with the ability to bypass overlay routing in LAN communications, (2) support for static and dynamic address allocation in conjunction with virtual nameservers through a distributed data store, and (3) support for transparent migration of IP endpoints across wide-area networks.

The approach is validated by a prototype implementation which has been deployed in grid and cloud environments. We present both a quantitative and qualitative discussion of our findings.

1. INTRODUCTION

The implementation and use of overlay networks (ONs[1]) as Virtual Private Networks (VPNs) for distributed computing has been explored in previous research [11, 30, 15, 29, 17, 18]. ONs enable users to easily merge resources into a homogeneous network system, providing connectivity across nodes distributed over multiple domains as if they belonged to a LAN. ONs also make the process of configuring the security of network environments significantly easier. By decoupling the public and private networks, the administrator of an ON need not be the same as the administrator of the physical network where the ON runs. This allows different security policies for the differing networks. For example, a public connection may be accessible by anyone but limit certain types of traffic, whereas an ON can be made accessible to only those with proper credentials and allow all forms of traffic.

However, current ON approaches lack the ability to autonomically configure a system consisting of both clusters and workstations in a scalable manner. Previous work on this topic [11, 13], analyzed the use of Peer-to-Peer (P2P) VN interfaces and the prospects of using dynamic host configuration protocol (DHCP) and address lookup through the use of a distributed hash table (DHT). However, a limitation of this approach was the need to incur ON routing overhead for all VN packets, including those destined to LAN hosts. The contributions made in this paper extend upon P2P-based overlay techniques [11, 13, 32] to support a unified framework where WAN ON routing can be reconciled with direct point-to-point LAN communication without sacrificing the self-configuration, scalability and fault tolerance properties of a P2P overlay. Specifically, we present a P2P-based VN which is novel in its ability to integrate the following features:

- Integrated support for three self-configuring methods to connect to the VN: (1) local-host ON router for a single VN endpoint, (2) single ON router for a cluster of VN endpoints, and (3) a novel hybrid model with a local-host ON router which can be bypassed for intra-LAN communications
- A single layer 3 subnet across multiple domains using an Ethernet layer gateway instead of IP layer gateway
- Machine addresses can be either automatically or statically configured
- Scalable address and name resolution
- Transparent, self-configuring WAN migration of VN endpoints

Specifically, the contributions that separate this paper from our previous work include the architectures of ON router and hybrid models allowing machines on a LAN to talk directly with each other avoiding VN software overheads, integration with standard networking protocols (ARP, DHCP, DNS) that provide a portable VN solution, and support for IP migration between VPN routers. Figure 1 depicts how the three deployment models considered in this paper relate to the P2P overlay architecture. Furthermore, we evaluate an implementation of the proposed approach qualitatively and quantitatively in environments that include distributed cloud resources.

An interesting new opportunity for virtual networking deployment exists in the realm of cloud computing, specifically infrastructure as a service (IaaS). IaaS provides an avenue

[1]Throughout this paper, the terms overlay network (ON), virtual network (VN), and virtual private network (VPN) will be used interchangeably. In this context a VPN/VN runs on top of an ON providing features such as IP and Ethernet to the ON.

SC09 November 14-20, 2009, Portland, Oregon, USA.

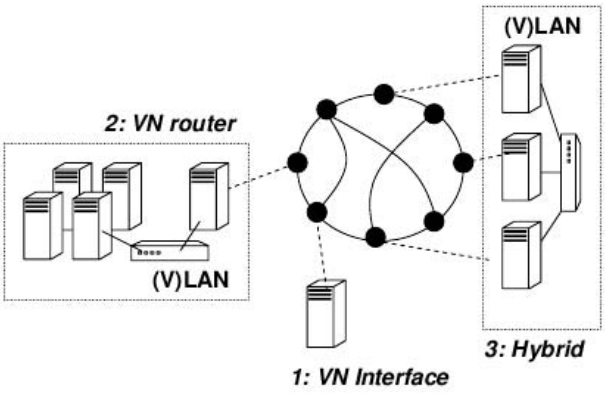


Figure 1: Illustration of the three different deployment models considered in this paper. In VN interface mode (1), each node has an overlay ID and communicates to all other nodes through VN tunneling. In VN router mode (2), only the router has an overlay ID and routes for a set of resources; LAN communication does not require VN tunneling. In hybrid mode (3), each host has an overlay ID; LAN communication does not require VN tunneling.

for users to instantiate systems on demand using remotely managed resources. This allows the user to focus on software in the system, thereby reducing or removing the cost of managing hardware infrastructures. In cloud environments, VNs provide a single private network that allows the distribution of resources amongst multiple cloud vendors, as well as seamless connection to local machines. The ability to deploy a VN across multiple providers, with the potential for supporting transparent migration, helps avoid vendor and data lock-in, which have been recognized as key challenges to the adoption of the cloud model [3]. In this context, we qualitatively evaluate the ability of our overlay to integrate with two of today's leading cloud providers (Amazon and GoGrid), demonstrating the ability to create a VN spanning two remote cloud infrastructures, local resources, and an overlay bootstrap deployed over hundreds of nodes distributed over PlanetLab [24].

This paper's organization follows. Section 2 provides the necessary background to understand the framework for our contributions by reviewing related and previous work. Section 3 introduces our VN models from a high level systems point of view. Section 4 describes the networking concepts used to implement a VN router, interface, and hybrid. Section 5 presents and evaluates our migration strategy. Section 6 qualitatively describes deployment and quantitatively evaluates VN routers and interfaces in cloud environments. Section 7 concludes the paper.

2. BACKGROUND

Three important areas that concern distributed systems are connectivity, security, and growth. In the field of grid systems, traditional environments consist of aggregate clusters, where each cluster has a head node that provides access and enforces security. However, the details of implementing and maintaining this environment, specifically the connectivity between resources and the head node as well as growth are left to the system administrator. To allow different clusters to merge into a unified system, administrators would have to place all resources on mutually accessible networks or create complex pathways between the two domains, creating potential security issues, limiting growth, and reducing the independence of that cluster. A popular solution to this problem involves the use of middleware brokering agents, such as Globus [10], which provide secure connections between clusters by requiring that each participating cluster's head node has a signed certificate and be publicly accessible to the brokering agent. This approach allows many small clusters to be aggregated together to form a distributed grid provided that the used cluster management software [2, 22, 6, 28] supports communicating with the brokering agent.

Cloud computing [3, 19], namely Infrastructure as a Service (IaaS), offers a new opportunity for distributed computing. IaaS allows users to focus on software and relegate the maintenance of hardware to cloud providers. Cloud providers offer features such as wide-area distribution of resources for fault tolerance, persistent storage, back-end private networking, and public network interfaces. Unlike grid environments, there has not been much discussion on cloud interoperability and user resources interacting with clouds, but interoperability is recognized as a hindrance to adoption. Efforts towards defining open clouds are beginning to appear with initiatives like the Open Cloud Manifesto [1]. In the context of our work and in its first revision, the Manifesto does not define the networking expectations of a cloud infrastructure.

In grids, not all scenarios require a layering approach to support multiple administrative domains provided by hosting one's own cluster. A scenario that is mutually beneficial for clouds exists in an environment where all-to-all connectivity is available for distinct and distributed resources. Virtual networking provides a framework enabling all-to-all network connectivity while retaining isolation from the physical network to facilitate the enforcement of security policies. This form of virtual networking is similar to Virtual Private Networking (VPN) services (e.g. OpenVPN [33] and CiscoVPN [5]) where all participants are in the same layer 3 network and thus have all-to-all connectivity. Bandwidth constraints, complex configuration/management, latency, and reliance on a single site are issues presented by traditional VPN configurations and led to the desire for better communication pathways provided by distributed overlay networks to support such infrastructures.

2.1 Related Work

Related work in the field of virtual networking on top of overlay networks for distributed computing includes Violin [15], VNET [29], ViNe [30], SoftUDC VNET [18], OCALA [17], N2N [8], and IPOP [11]. The primary feature found in all virtual networking software is the support for all-to-all native IP communication amongst peers in the virtual network, though their mechanisms for supporting this are different. Table 1 summarizes key differences amongst these approaches, which have been motivated by different assumptions about the target environment and use. The contributions of this paper largely stem from and extend initial work done in the IPOP overlay as described in [11, 13], which is detailed in the following subsection.

2.2 IPOP: P2P Virtual Networking

	Overlay	Routing	Configuration	Miscellaneous
IPOP	Structured P2P overlay with O(Log N) routing hops, where N is the size of P2P network. Self-optimizing shortcuts and STUN-based NAT traversal.	Mapping stored in DHT resolves virtual IP address to P2P address. Virtual network packets are routed to corresponding P2P address.	Each machine runs P2P VPN software with a dynamic IP address in a common subnet. Common configuration shared amongst all hosts.	Supports encrypted P2P links and end-to-end VPN tunnels (unpublished work). Migration possible; routes self-configure without user intervention, product of the P2P overlay.
N2N	Unstructured P2P network, super nodes provide control paths, forms direct connections for data.	Broadcast for discovery and overlay for control. No organization, no guarantees about routing time.	Requires N2N software at each host, must connect to a super node. Supports layer 2 Ethernet network.	Supports shared secrets to create private tunnels between edges. Migration not discussed, but potentially free due to layer 2 approach.
OCALA	Not tied to any specific overlay, layer 3 middleware.	Based upon chosen overlay.	Requires OCALA stack, overlay configuration, and IP to overlay mapping.	Security is overlay based or SSH tunnels. Migration not mentioned.
SoftUDC VNET	Decentralized with explicitly configured overlay routes.	Broadcast for discovery.	Requires software on each host and one proxy per site. Layer 2 networking.	Security is not discussed nor is wide-area migration.
ViNe	ViNe authority configures global network descriptor table (GNDT) explicitly at each router. Supports proxying to one location through another and NAT traversal.	GNDT provides overlay routes for all routers in overlay.	Each subnet is allocated a single router. Each host must be configured for regular and ViNe networks, but no VN software needed on host.	Supports encrypted tunnels between ViNE routers, migration not discussed.
Violin	Decentralized network with statically configured overlay routes.	Broadcast discovery for Ethernet, static routes for IP subnet.	Virtual hosts connect VMs to the VN. Hosts connect to virtual switches or proxies (gateways). Switches connect to proxies. Sites are typically allocated an IP address space.	Security potentially through the use of SSH Tunnels. Migration possible; requires reconfiguration of switches.
Virtuoso VNET	Decentralized with explicitly configured overlay routes.	Broadcast for discovery. Bridging learns paths after initial discovery. Virtual network packets are routed between VNET proxies. Can be configured manually.	Each site runs a proxy providing Ethernet bridge to other proxies. VM hosts forward packets to local proxy. Proxies configured to connect to other proxies.	Security through the use of SSL and SSH Tunnels. Layer 3 migration, product of layer 2 virtualization.

Table 1: Virtual Network Comparison

IPOP uniquely differentiates itself from related virtual network techniques in how it builds upon a Peer-to-Peer (P2P) overlay for virtual IP packet routing and for distributed hash table (DHT)-based object storage/lookup. In doing so, it removes the requirement of having any centralized hosting mechanism. Furthermore, a P2P system can provide features that allow two private-addressed machines constrained by distinct NATs (Network Address Translation devices) to communicate with each other through traversal techniques as well as through the P2P overlay without configuring any routing rules, similar to the proxying techniques used in other VNs. This section discusses the features of P2P in IPOP that reduce the complexity in configuring a VN.

In a P2P system, each node has two ways to be addressed: via the lower layer network, e.g. an IP address, and through the overlay via a P2P address. The machine's IP address is only used to help bootstrap overlay connections; the P2P address is then used when communicating through the overlay. As described previously [13], IPOP introduced two concepts that allowed namespaces and multiple virtual IP networks to share the same P2P system, and the use of a method for dynamic allocation of IP addresses. In this method, the namespace and the virtual IP are hashed to create a key for lookup of the P2P address.

Supporting distributed address allocation and lookup requires access to a distributed data share that supports atomic and idempotent writes. To this end, IPOP uses a DHT provided by the underlying P2P system. A DHT is similar to a hash table: a database where data is stored in a (key, value) format. During address allocation, a machine attempts to perform an atomic write where the DHT key is the namespace and requested IP address and the value is its P2P address. The DHT is also used to store information about the namespaces that is used during DHCP configuration, such as the valid address range, lease time, and reserved IP addresses.

Furthermore, because IPOP uses the P2P address for message routing, which is decoupled from the physical network address of a host, virtual IP address migration happens transparently to applications (even across domains), and connectivity can be established as quickly as it takes for P2P links to be reestablished.

As mentioned in Table 1, IPOP handles only layer 3 virtualization and has limited interaction with layer 2. As such,

prior to the techniques described in this paper, IPOP could only support a one-to-one mapping of IPOP instance to host.

3. VIRTUAL NETWORK MODELS

Traditionally, VNs provide either an interface model, where there exists one VN software stack per host, or the router model, where there exists one VN software stack per some subset of machines in a LAN. In this section, we present a comparison of qualitative advantages and disadvantages of the two approaches. We also present a new model, hybrid, that bridges the gap between the two models by taking a subset of features from both. Graphical representations of the models can be found in Figures 2, 3, 4. In this section, we focus on the basic architecture of the VNs leaving the system independent networking features for the following section.

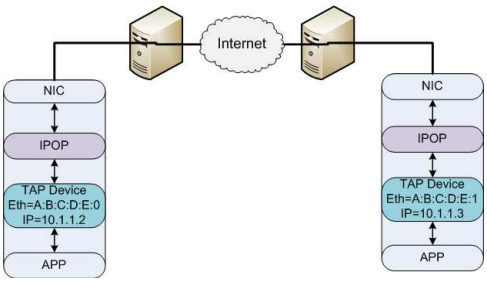


Figure 2: A VN deployed as an interface for single machine usage. The user of the machine is presented two interfaces on two different IP subnets. All non-VN subnet based traffic is routed normally via the default interface.

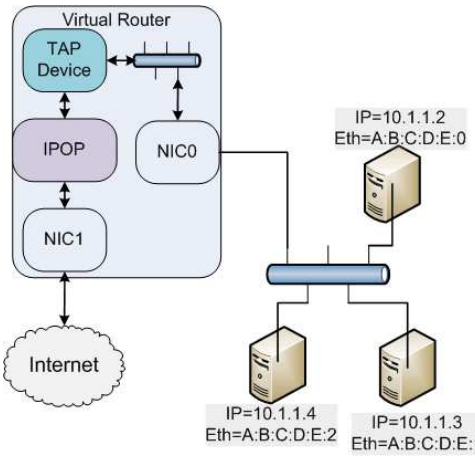


Figure 3: A VN deployed as a router providing virtual network access for an entire layer 2 network. Each machine in the network only has a VN-based address, though they can communicate directly with each other (and with proper routing rules and NAT setup the Internet as well). The machine hosting the VN can also have an IP address in the network by assigning one to the bridge.

The prime factor which the three designs share is the TAP [21] device, a Virtual Ethernet device that is available

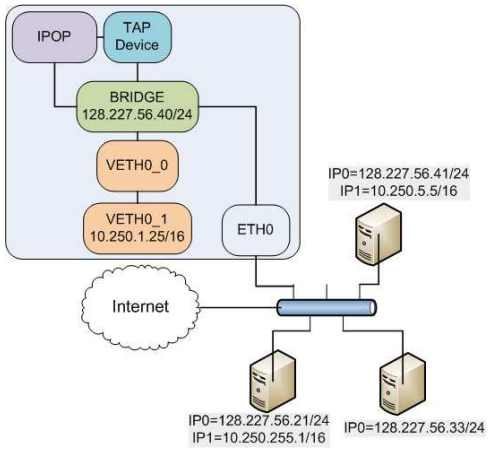


Figure 4: A VN deployed in a hybrid mode providing virtual network access for a single machine but bypassing the VN when a VN peer is local. This model is similar to having two network cards from a single machine going to one switch. The key feature is that this model allows a machine to be in multiple IP address space subnets and have layer 2 traffic as well.

for almost all modern operating systems, including Windows, Linux, Mac OS/X, BSD, and Solaris. The TAP device functions by letting software write incoming packets to the device which are treated as packets that came from over the network. Packets that are written to the TAP device by O/S sockets are available by reading the TAP device. The VN receives incoming packets from the overlay and writes them to the TAP device, and receives outgoing packets from the TAP device and sends them through the overlay.

The added feature of the router model (Figure 3) is that the TAP device is now bridged to the entire network. In other words, the TAP device virtualizes a bridge to other physical networks. Thus packets can be routed to it and sent through the overlay. Packets received by the VN can be written to the TAP device with the destination being a machine physically connected to the machine hosting the VN router. This approach eliminates overhead between two machines in the same physical network as they can talk directly with each other without any VN middleware. Furthermore, the router can easily be deployed in existing systems, simplifying the migration to virtual networking. This model suffers from having a single point of failure for a LAN (namely, the VN router). This is a situation that does not occur in the interface mode. Nonetheless, depending on deployment scenarios, this VN model reduces overall resource utilization, since each machine using the VN router would not have to deal with network virtualization as that is dedicated to a specific VM.

Finally, we present the VN hybrid model (Figure 4) which takes from the VN router the ability to allow machines on the same LAN to communicate directly with each other without VN overhead while still providing the fault tolerance that the VN interface provides. Assuming that there is no overhead in the O/S' pseudo-drivers, the performance should be identical in the LAN to the VN router and to the VN interface for WAN. From a high level point of view, the piece that makes this possible is the VETH pseudo device

that provides a virtual Ethernet pair. This allows users to add additional network cards to a bridge and thus allow multiple IP addresses in this environment. The reason for this lies in the nature of the state of the interfaces connected to the bridge. Devices directly connected to the bridge go into promiscuous mode, that is, all packets sent to them are forwarded on as if they are a wire. In non-promiscuous mode, the network card will drop packets that are not destined for that network card. So in that case, it is not possible to assign more than one IP address to a bridge, because it and all devices connected to it are viewed as one big network interface. By connecting the VETH device, we are able to uniquely identify Ethernet addresses and thus additional IP addresses. In contrast, aliasing a Ethernet card only provide additional IP addresses and services that rely on layer 2 networking. In this case, some services may not work, for example, DHCP does not work on aliased network cards.

To summarize, Table 2 presents a qualitative comparison of the three deployment models.

4. IMPLEMENTATION

A self-configuring system must be transparent to the environment where it resides and require no interaction from the administrator besides starting the system. For VNs, it means the ability to join a network without explicitly adding network rules or routing tables. In this section, we present the methods used in layer 2 and layer 3 networking that allow the deployment of self-configuring VNs alongside an existing system. Specifically, we dissect Figure 5 and describe the sections in terms of the networking protocols and features, namely:

- ARP, Address Resolution Protocol, to allow communication in a LAN
- DHCP, Dynamic Host Configuration Protocol, for dynamic address allocation
- IP, Internet Protocol, for static addresses
- DNS, Domain Name Servers, for virtualized name services

4.1 Layer 3 Communication in a LAN

IP is a layer 3 protocol. Layer 2 devices such as switches, bridges, and hubs are not aware of IP addresses. When a layer 3 packet is being sent on a layer 2 network, the address resolution protocol (ARP) is used to determine the layer 2 address. This process, as shown in Figure 6, begins by the sending of a layer 2 broadcast message which contains an ARP request, requesting that a node owning the target IP address respond to the sender of this packet. If a node owns the target IP address, they respond with an ARP reply, making themselves the sender and the original sender is the message recipient. The Ethernet header consists of the source address being the sender and the target being the destination. By listening to these requests, layer 2 devices such as a switch can autonomously learn the location of nodes holding Ethernet addresses are and can forward packets through appropriate ports as opposed to broadcasting or dropping them.

In a typical IP subnet, all machines talk directly with each other through switches. As such, they must learn each other's Ethernet address. In our virtual network model, we aim to provide a large, flat subnet spanning across all nodes connected to the VN. To accomplish this, the VN provides the ability to virtualize a bridge, similar to proxy ARPs [25] used to implement a transparent subnet gateway [4]. In this

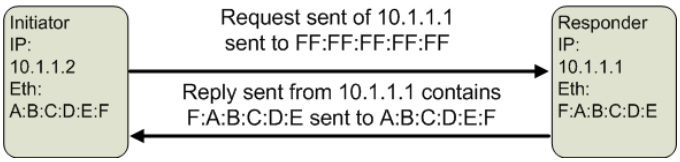


Figure 6: ARP Request/Reply Interaction.

scenario, the VN would need to respond to the ARP packets with a fake layer 2 address. Layer 2 devices in the system would then route all packets destined for that layer 2 address to the VN.

In our model (Figure 5), ARPs are only responded to if (a) they are inquiring about a VN IP address, (b) the VN address is not locally allocated, and (c) there is a P2P:IP mapping. If all those are true, then an ARP response is sent back to the sender. ARPs are occasionally sent out during the course of communication and thus if a machine migrates to a VN router, the VN router will no longer respond with ARPs. An ARP response sent by the VN requires a source Ethernet address, bridges and switches will see the response and will forward all traffic towards the TAP device for that Ethernet address. A VN device can use the same Ethernet address for remote entities.

Prior to the introduction of the VN hybrid, our systems used the Ethernet address FE:FD:00:00:00:00 to refer to remote entities. If each VN hybrid used this address, there would be layer 2 collision causing a single hybrid to have all traffic sent to it. In hybrid mode, each VN must generate a unique "remote" Ethernet address at run time. Our experience and research has led us to the following solution: (1) use FE:FD for the first two bytes as they tend to be unallocated and (2) assign random values to the 4 remaining bytes. Applying the birthday problem in this context, the expected probability of address collisions is small for typical LAN environments (less than 50% if the average number of VN hybrid nodes on the same L2 network is 65,000).

4.2 Allocating Addresses

If a packet is an IP packet, the VN has a new decision to make: whether or not it should attempt to route the packet. This decision is made by checking 1) does the VN own the address and 2) can the VN route the packet. So we begin by discussing methods of how a VN may own an address and then discuss how to determine IP packet routing.

IP addresses are traditionally allocated in one of three ways: 1) statically, 2) dynamically through DHCP, or 3) through pseudo-random link-local addressing. In our model, we focus on static and dynamic addressing.

DHCP as defined in the RFCs [9, 27] enables dynamic configuration of addresses, routing, and other networking related features. While many different client and servers exist, they all tend to support the basic features of allowing the server to specify to a client and IP address, a gateway address, and domain name servers. As shown in Figure 7, the steps in DHCP are:

1. Client sends Discover packet requesting address.
2. Server receives the packet, allocates an address, and sends an Offer of the address and other network configuration.
3. Client receives and acknowledges the Offer by sending a Request message to accept the Offer.

	Interface	Router	Hybrid
Host LAN	No assumption	Ideally, VLAN	No assumption, though may have duplicate address allocation in the same subnet for different namespaces.[2]
Host software	IPOP, tap	End node: none. Router: IPOP, tap, bridge	IPOP, tap, veth, bridge
Host overhead	CPU, memory	End node: none. Router: CPU, memory	CPU, memory
LAN traffic	Through IPOP	Bypasses IPOP	
Migration	Handled by node	Involves source and target routers	Handled by node
Tolerance to faults	Nodes are independent	Router fault affects all LAN nodes	Nodes are independent

Table 2: Qualitative comparison of the three deployment models

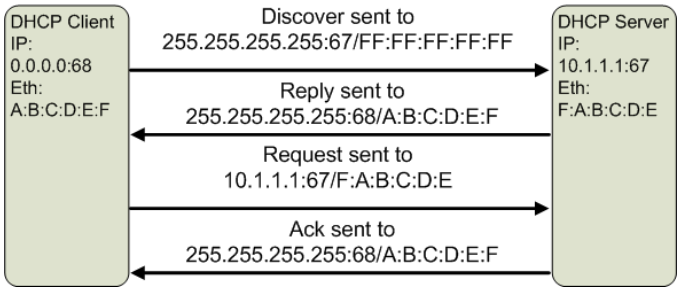


Figure 7: DHCP Client/Server Interaction.

4. Server receives Request message and returns an Ack message containing the same details as the Offer.

During the DHCP phase, the VN will communicate with the overlay and allocate an address for the requester. Similarly, a VN model can review packets coming into the VN, review the sender IP address, and request from the overlay that the address be allocated to the VN. Thus to the overlay, static addressing is identical to dynamic addressing. There is one caveat with static addressing: it is difficult to handle address conflicts. In our model, this is done by the overlay dropping address allocation requests made for a previously allocated address. DHCP does support requesting an address and thus one could argue that this method is superior to using static addresses. Another key feature of DHCP is that it provides network information, namely in our case, DNS server addresses.

If an overlay allocates an address to the VN, then the VN owns it. The other address that the VN owns is the null address, 0.0.0.0, which is sent during DHCP to indicate that the machine has no address prior to the request.

Routing is contingent on the ability of the overlay to an address and whether or not it supports broadcast and multicast packets. In our models, we implemented broadcast and multicast by having all members of a subnet associate on the DHT a value (the overlay address) to a specific key, i.e., namespace:broadcast. Then when such a packet is received, it is sent to all addresses associated with that key. Then it is up to the VN at each site to filter the packet. This is sufficient to support deployments where multicast or broadcast is not relied upon extensively; we consider support for scalable multicast as a topic for future work.

[2]A namespace provides a unique global address space inside a VN. The filtering for namespace occurs in the VN software and bypassing that software, as in the case of the hybrid, can result in undesirable behavior.

4.3 Domain Name Servers and Services

Name services allow machines to be addressed with names that are more meaningful to users than numeric addresses. Certain applications and services require domain name checking, such as Condor. To support DNS, this requires that the OS be programmed with the VN's DNS servers IP, which we take generically to be the lowest available IP address in a subnet. In static configuration, this process requires the user to manually add this address, though through DHCP this is set automatically.

In the state representation of the VN (Figure 5), the VN checks the IP packet to ensure that the destination IP and port match that of the virtual DNS server and the well-known DNS port, 53. In the event of a match, the packet is passed to the VN's handler for domain names. From our experience, there are two common uses for names, 1) because applications require it, and 2) to assist users in finding resources. For 1), we have implemented a system that takes in an IP address and maps it directly to a name, such as, 10.250.5.5 maps to C250005005. For 2), we take advantage of the DHT and perform a Put in the key namespace:hostname with the value being the mapped IP address.

4.4 Hybrid Configuration

The key difference from the hybrid and router models is that the hybrid model routes for only a single network device, say A, and thus must ignore messages that do not originate from A. This has other ramifications, strictly speaking, the hybrid model does not know about the existence of all machines in a local area network, because it does not own them. So when an ARP request of some remote machine, say B, is sent by A, the hybrid interface must receive the result. For this, we suggest sending a similar message to B requesting the local Ethernet address with the requester being the same pseudo-entity used for the transparent subnet gateway. If no message is returned after a set amount of time (we use 2 seconds) then the original ARP can be responded to with the pseudo-entity being the target.

To determine which Ethernet the hybrid routes, it can be given the name of the device at startup and query the operating system for the Ethernet address. Other potentially interesting approach involve the use of a few hybrid nodes to route for many LAN machines, akin to the topic discussed

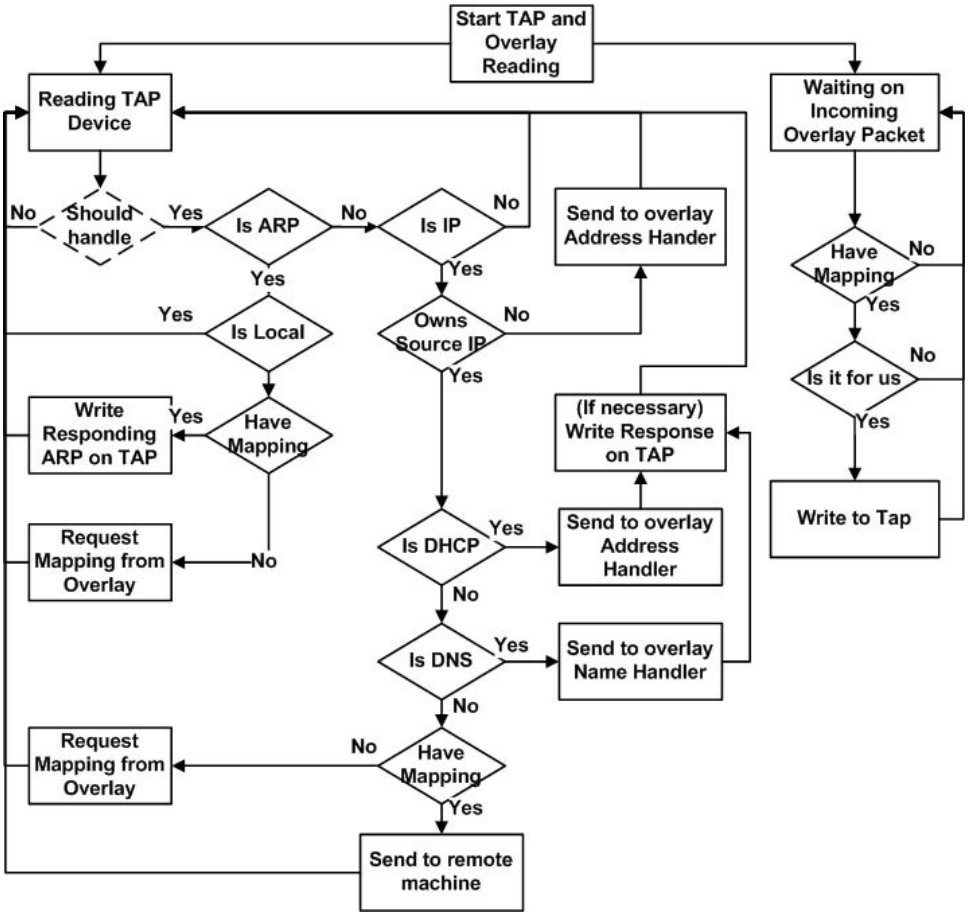


Figure 5: The state diagram of our proposed VN. In this model, a VN interface is identical to a VN router with the caveat that the TAP device is not bridged, thus isolating the VN traffic. The "Should Handle" with dashed lines is a feature that is specific to the VN hybrid; that is, a VN hybrid must be configured to communicate for a single network device.

in [32] regarding a few routers for many machines, though this is left as future work.

5. SUPPORTING MIGRATION

Apart from advantages like performance isolation, security, and portability, one of the significant advantages of using VMs is the capability to migrate the VM with its entire software stack from one physical host to another. This migration may be performed in a stop-restart manner, where the VM is paused, migrated to another host and restarted, or in a live mode, which attempts to minimize down time to reduce interruption of services running on the VM.

VMs including Xen [20], VMware ESX [23] and KVM [26] support migration with two critical requirements: (1) file systems (disk images) must be on a shared storage system (i.e. network file systems or storage area networks) and (2) to maintain network connectivity, the migration must occur within an IP subnet. In order to retain network connectivity after migration, the VMM must notify the LAN of the VM's new location. The new VMM host generates an unsolicited ARP reply which broadcasts to the entire network the VM's new location.

The VN interface and hybrid models support migration of the virtual address using techniques previously described in [11]. This is a product of the decentralized, isolated overlay approach where each overlay end point has a one-to-one mapping to VN end point, e.g., P2P to IP. When a VN interface or hybrid model migrates, the overlay software must reconnect to the overlay, at which point, packets will begin to be routed to the VN endpoint again, completing migration.

Unlike interface and hybrid models, the VN router does not support a one-to-one mapping. In fact, a VN router tends to have one P2P address for many IP addresses. When a machine with a VN IP wants to migrate, it cannot also take its P2P address with it otherwise it would end connectivity for the rest of the members of the VN router shared overlay end point. A solution to this problems requires the ability to delete IP-to-P2P mappings in the DHT, detect new addresses on the network, and inform senders that an IP is no longer located at that overlay end point. With these capabilities, transparent migration can be achieved for the VN router model as follows.

The VMM initiates a migration on a source node. Until the migration completes, the VN router at the source continues to route virtual IP packets for the VM. Upon completion

of migration, the VN router at the target learns about the presence of the migrated VM by either receipt of an unsolicited ARP or by proactively issuing periodic broadcast ICMP messages on its LAN. The VN router attempts to store (Put) the IP:P2P address mapping in the DHT, and queries for the existence of other IP:P2P mapping(s). If no previous mappings are found, the VN router assumes responsibility for the IP address. Otherwise, the VN router sends a migration request to each P2P address returned by the DHT. The VN router receiving a migration request confirms the existence of the IP address in its routing table and that if there is that there is no response to ARP requests sent to the IP address. If these conditions hold, it deletes its IP:P2P mapping from the DHT and returns true to the migration request; otherwise, it returns false. If the migration request returns true, the VN router at the target LAN starts routing for the virtual IP address; if it returns, false, the VN router does not route for the virtual IP address until the previous IP:P2P mapping expires from the DHT.

In addition to VN routers synchronizing ownership of the migrated virtual IP address, any host that is connected to that machine must be informed of the new P2P host. Over time, this will happen naturally as ARP cache entries expire and the IP:P2P mapping is looked up from the DHT. Additionally, the VN router at the source may keep forwarding rules for the migrated IP address for a certain period of time, akin to mobile IP but not on a permanent basis. A more direct approach, as implemented in our prototype, involves the VN router notifying the connected host of a change in ownership, resulting in the host querying the DHT for the updated P2P end point. An evaluation of tradeoffs in the migration design, while interesting, is outside the scope of this paper.

6. EVALUATION

In this section, we present an evaluation of the different VN models, using prototype implementations built upon IPOP. In the Grid evaluation, we simulate a client/server environment and investigate CPU / networking overheads related with each approach. We also present results with cloud deployments: an evaluation of a proof of concept internetworking of multiple clouds and local resources, and evaluation of the overhead of the different approaches in WAN and LAN environments. In all WAN experiments, a wide-area IPOP overlay network with approximately 500 overlay nodes distributed across the world on PlanetLab resources is used to bootstrap VN connections and to support DHT-based storage and P2P messaging.

6.1 Tools

The proposed VN models place varying demands on the resources of the systems involved. We chose to focus on CPU as our experience suggests that this is the most significant limiting factor. As we will present, the CPU load offered by these models depends on the bandwidth of the underlying network link, since a larger bandwidth requires more processing of packets.

To evaluate these VN models, we employed Netperf [16] and SPECjbb [7]. These tools were used as follows:

Netperf is used to estimate the latency and bandwidth of the different VN models. The latency is measured by deploying Netperf in the TCP_RR mode, which measures the number of 1-byte request-receive transactions that can be completed in a second. The bandwidh is estimated by running Netperf in the TCP_STREAM mode, which is a bulk transfer mode. It should be noted that in situations where the link bandwidths were asymetric, we deployed Netperf in both directions. Since both the latency and bandwidth are dependent on the CPU available, we ran Netperf in situations where it was the only active workload (referred as "no spec" in Figures), and in situations where there was a CPU intensive benchmark running on the system.

SPECjbb simulates a three-tier web application with all the clients, the middle tier, and the database running on a single system in a single address space (inside a JVM). On completion, the benchmark provides the metric in terms of business of operations per second (bops). The bops score of the system under test depends on both the CPU and the memory in the system, as the entire database for the benchmark is held in memory. This benchmark generates negligible disk activity and no network activity. Thus, by comparing the bops value when SPECjbb is run with and without the Netperf (with different VN models), we get an insight into the CPU load offered by the VN model. SPECjbb was configured to run for 4 minutes with 1 warehouse and a JVM heap size of 256 MB. The running time of Netperf was also set at 4 minutes.

6.2 On the Grid

Our initial evaluation involves testing a client-server environment. Our baseline hardware consisted of quad-core 2.3GHz 5140 Xeon with 5 GB memory and Gigabit network connectivity. Each VM was allocated 512 MB of RAM and ran Debian 4.0 using a Linux 2.6.25 kernel. The client side consisted of 4 VMs on 5 machines. The server side consisted of 5 VMs on one machine with 4 acting as servers and 1 acting as a gateway, which was necessary to control bandwidth into the system, done through the Linux utility **tc** [14], traffic control. In this environment, each server had 5 clients communicating with it. The setup is shown in Figure 8.

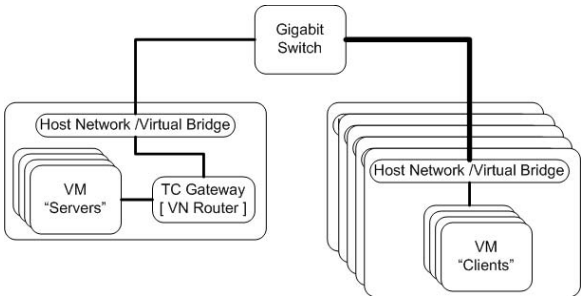
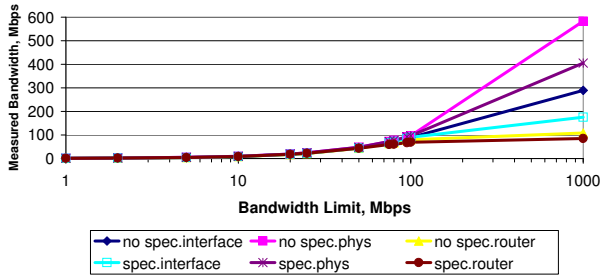
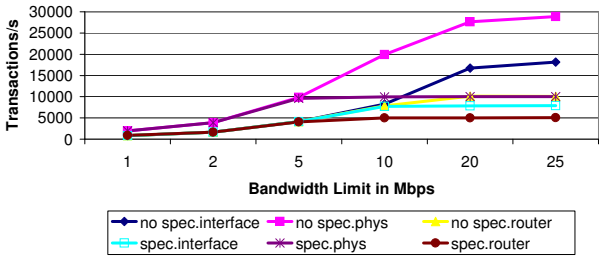


Figure 8: The system setup for our grid experiments. The VM "Servers" ran SPECjbb and were also the site for the collection of the netperf benchmarks. All the VM "Servers" were connected through the TC Gateway through host-only networking to the VM "Clients". All traffic for the VM "Servers" passes through the TC Gateway, which also doubled as the Router in the Router experiments.

The evaluation results are presented in Figure 9 and 10. The maximum bandwidth of 600 Mbps is achieved when neither virtual network nor traffic shaping are enabled ("no spec.phys" at 1000 Mbps limit in Figure 9(a)), which is only 60% of the theoretical maximum. We attribute this limit

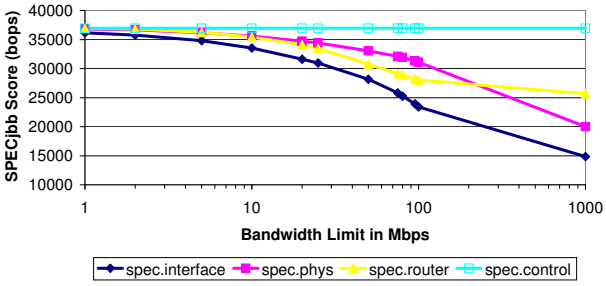


(a) TCP Stream bandwidth, SPECJbb load

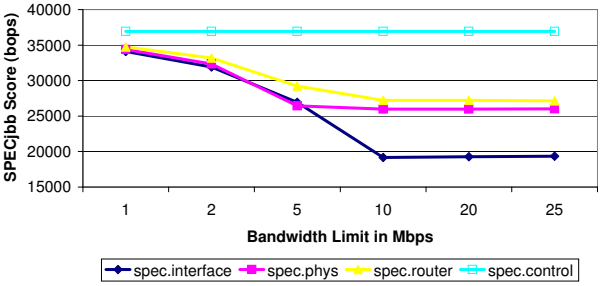


(b) TCP RR latency, SPECJbb load

Figure 9: Netperf measurements with and without SPECjbb load. Lines are of the form (no spec, spec).(phys, interface, router). Where "spec" indicates SPECjbb benchmark is active, while "no spec" indicates that SPECJbb is inactive. "phys" implies the absence of IPOP with benchmarks occurring directly over the "physical" network card. "interface" and "router" present the results for VN interface and router models respectively.



(a) SPECJbb score, TCP Stream load



(b) SPECJbb score, TCP RR load

Figure 10: SPECjbb scores with and without Netperf load. Lines are of the form spec.(control, phys, interface, router). "spec" implies that SPECJbb executes in all tests. In "control" Netperf is inactive, that is, it is the maximum attainable value for SPECJbb. "phys" implies the absence of IPOP with benchmarks occurring directly over the "physical" network card. "interface" and "router" present the results for VN interface and router models respectively.

to the cost of VMMs, specifically the time required for a packet to traverse both VMMs networking stack as well as the hosts networking stack. Another observation was that transactions per second (Figure 10 (b)) do not improve significantly for **tc** bandwidth limit above 25 Mbps in all cases; thus we focus on the relevant data up to this limit.

Distinguishing features of the different VN models include the following. Figure 9(a) shows that bandwidth in all VN models is comparable with traffic control limit up to 75 Mbps. Beyond this point, the interface model achieves better bandwidth than the router model (VN processing is distributed across multiple processes); the spec/no spec ratio in the router model is smaller than in the interface model because there is less resource contention caused by VN processing on end nodes. For the same reason, the router tends to achieve better SPEC results (Figure 10(a)) than the interface. Figure 9(b) shows that the router performs poorly compared to the interface model in terms of transactions/second, though it achieves a better ratio of SPECjbb score (Figure 10(b)) to transactions than the interface at constrained bandwidths (less than 5 Mbps).

The hybrid method was tested, and results were nearly identical to those of the interface, from the point of view of the WAN part of the VN, it is the same architecture. These results are not reported in the plots as they add little value and further obfuscate the results.

The bandwidth cap observed in the router approach reflects the performance achieved by the current prototype of the router, subject to VM overheads. The use of VM is an assumption that is valid in the domain of cloud computing where all resources run in a VM. In this experiment, we focused on the interplay between resource consumption by overlay routers and application performance. Optimized user-level overlay routers running on dedicated physical machines have been reported to achieve performance near Gbit/s in related work [31]; we expect that such cap can be raised substantially with improved VM handling of network packets and user-level router optimizations.

6.3 In the Clouds

The goal of this experiment is to demonstrate the feasibility of connecting multiple cloud providers as well as local resources together through virtual networking. The sites chosen for evaluation were local resources at Universty of Florida and cloud resources provided by Amazon EC2 and GoGrid. A qualitative observation here was that the differences in the networking infrastructure exposed by different cloud providers reinforce the importance of the virtual network to allow flexibility in how end nodes are connected.

	EC2 / UF	EC2 / GoGrid	UF / GoGrid
Stream Phys	89.21	35.93	30.17
Stream VN	75.31	19.21	25.65
RR Phys	13.35	11.09	9.97
RR VN	13.33	10.69	9.76

Table 3: WAN Results for inter-cloud networking. Stream is in Mbs and RR is in trans/s (The inverse of trans/s would be equal to the average latency).

	VN Interface	VN Router	VN Hybrid	Physical
Stream	109	325	324	327
RR	1863	2277	2253	3121

Table 4: LAN results performed at GoGrid. Stream is in Mbs and RR is in trans/s. Interface and Physical used the eth0 NIC, while Router and Hybrid used eth1. Different VLANs may give different results.

Specific network configurations for the clouds were as follows:

- Amazon EC2 provides static IP networking (public and private), no Ethernet connectivity, and no ability to reconfigure IP addresses for network. Currently, only the VN interface model is supported.
- GoGrid provides 3 interfaces (one public, statically configured, and two private, which can be configured in any manner); the 2 private interfaces are on separate VLANs supporting Ethernet connectivity. The VN interface, router, and hybrid models are supported.

In this experiment, we have narrowed down the performance evaluation to focus on WAN and LAN performance of VNs in cloud environments and consider Netperf single client-server interactions only. Because Amazon only supports Interface mode, we only evaluate it in the WAN experiment. We have observed that, within Amazon, the VN is able to self-organize direct overlay connections [12]. Each test was run 5 times for 30 seconds, the standard deviation for all results was less than 1. Because of this, we only present the average in Table 3.

It can be seen in Table 3 that the VN adds little overhead in the Netperf-RR experiment. Between UF and GoGrid as well as between UF and Amazon EC2, the overhead for the Stream experiment was about 15%. This may be attributed to the additional per-packet overhead of the VN and the small MTU set for the VN interface (1200). The MTU, or maximum transmission unit, is the larget packet that is sent from an interface. IPOP conservatively limits the VN MTU to 1200 down from the default 1500 to allow for overlay headers and to work properly with poorly configured routers, which we have encountered in practical deployments. A more dynamic MTU, which will improve performance, is left as future work. The EC2 / GoGrid experiment had greater overhead which could possibly be attributed to by the VM encapsulation of cloud resources.

Table 4 shows that some of our performance expectations for the different models in a LAN were accurately predicted while others were not so clear. Stream results match the expectation that VN models hybrid and router bypass virtualization and get near physical speeds, whereas interface does not. Interestingly, RR had rather poor results for Router and Hybrid though further testing seems to indicate that this is an issue of using the VLAN connected network interfaces as opposed to the public network connected interface.

6.4 VN Router Migration Evaluation

For the analysis of our implementation, we setup two Xen-based VMware VMs co-located on the same quad-core Xeon 5335 2 GHz machine each with 1 GB memory allocated running only a minimal configuration with a SSH server. The goal of the evaluation was to determine overlay overheads and attempt to minimize the cost of the VM migration. The experiment, as shown in Figure 11, involved migrating a Xen guest VM between two Xen host VMMs running in VMware. Although they are hosted in the same infrastructure, the two domains are connected to two separate VLANs, and thus isolated. As mentioned earlier, resource information is stored in a DHT running on top of our ON which is deployed on PlanetLab. Thus the migration overheads in the experiment capture the cost of wide-area messaging in a realistic environment. During the course of the experiment, we tried over 50 different IP addresses migrating them about 10 times each in attempt to gain some insights in the cost of using the DHT with support for deletes and VN router messages as a means to implement migration. The result, presented in Figure 12 gathered from the experiment was how long the VN IP was offline, measured by means of ICMP ping packets. On average, the overhead of VN migration was 20 seconds. This overhead is in addition to the time taken to migrate a VM, since the VN routers begin to communicate only after migration finishes.

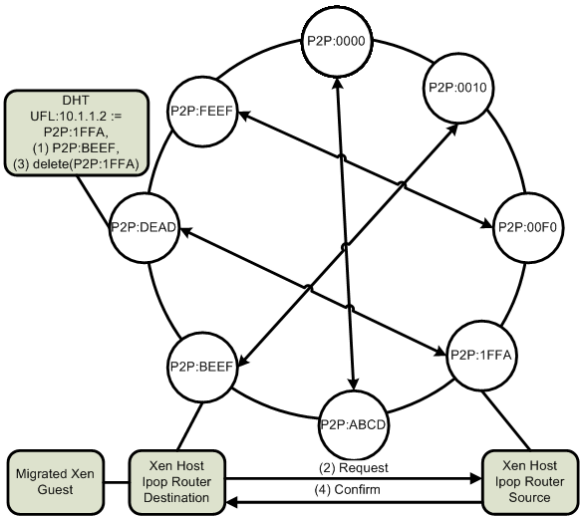


Figure 11: The VN operations that occur after a guest (VM) has been migrated. (1) The destination retrieves the P2P information of the source from the DHT and optimistically places its information into the Dht. (2) The destination requests that the source delete its information from the DHT. (3) The source confirms that the VM is no longer present and performs the delete. (4) The source notifies the destination that its request has finished successfully.

7. CONCLUSIONS

A key aspect of our contribution is the use of common network protocols as the building blocks for self-configuring,

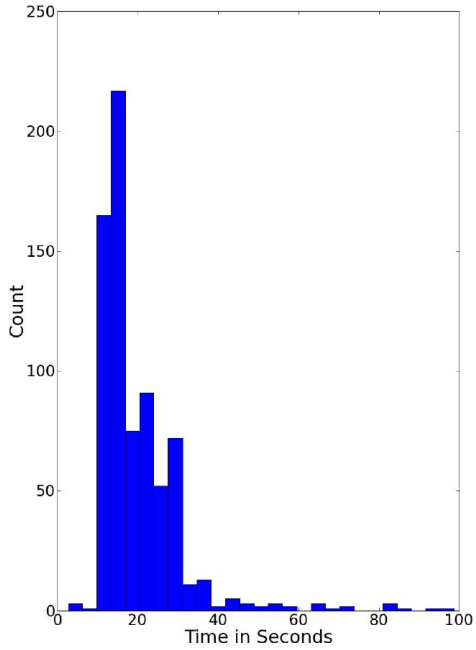


Figure 12: Over 50 different IPs migrated about 10 times each. The average was 20.11 seconds with a standard deviation of 10.89. In this experiment, the majority of this time comes from VN migration, whereas VM migration requires less than a second.

transparent VNs. The discussion focuses on address allocation and resolution through methods such as packet parsing, DHCP, ARP, and ICMP. Knowledge of these protocols helped in the formation of a novel VN model that supported key features of both router and interface VN models, bypassing overlay overheads in LANs while still providing fault tolerance by having each end point support its own VN software stack.

Through the qualitative and quantitative discussion presented in Sections 3 and 4, respectively, we have come to the following conclusions:

1. VN interface provides good isolation, best networking performance over WAN, and is the easiest to configure. However, it suffers LAN performance overheads, requires host software and incurs resource contention on end hosts.
2. VN router allows for quick bootstraping of an environment, allows for resource consolidation, and avoids overlay overheads in the LAN. However, it can be a bandwidth bottleneck in network-intensive applications and can be a single point of failure.
3. VN hybrid offers equivalent WAN performance of the interface and LAN performance equivalent to that of the router. However, it requires additional software at each VM and may cause potential address conflicts within a subnet.

Therefore, it is our opinion that no one model is the overall best. The decision on choosing which one is appropriate is largely a function of the configuration of the environment where it will be deployed. The flexibility of the proposed approach allows for a VN overlay to support all proposed VN deployment models and not imposing a single model to all resources.

8. ACKNOWLEDGMENTS

We thank our shepherd, Henrique Andrade, and the anonymous reviewers for their useful comments and feedback. This work is sponsored by the National Science Foundation under awards 0751112 and 0721867. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

9. REFERENCES

[1] Open cloud manifesto. `http://www.opencloudmanifesto.org/`, 4 2009.

[2] Altair. Pbs pro. `http://www.altair.com/software/pbspro.htm`, March 2007.

[3] M. Armbrust, A. Fox, R. GrifiňĄth, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical report, University of California at Berkeley, February 2009.

[4] S. Carl-Mitchell and J. S. Quarterman. Rfc 1027 - using arp to implement transparent subnet gateways, October 1987.

[5] Cisco. Cisco vpn. `http://www.cisco.com/en/US/products/sw/secursw/ps2308/index.html`, March 2007.

[6] P. Computing. Load sharing facility, lsf. `http://www.platform.com/`, March 2007.

[7] S. P. E. Corporation. Specjbb2005. `http://www.spec.org/jbb2005/`, 2005.

[8] L. Deri and R. Andrews. N2n: A layer two peer-to-peer vpn. In *AIMS '08: Proceedings of the 2nd international conference on Autonomous Infrastructure, Management and Security*, pages 53–64, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] R. Droms. *RFC 2131 Dynamic Host Configuration Protocol*, March 1997.

[10] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid - enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15:2001, 2001.

[11] A. Ganguly, A. Agrawal, O. P. Boykin, and R. Figueiredo. IP over P2P: Enabling self-configuring virtual IP networks for grid computing. In *International Parallel and Distributed Processing Symposium*, Apr 2006.

[12] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo. Wow: Self-organizing wide area overlay networks of virtual workstations. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 30–41, June 2006.

[13] A. Ganguly, D. Wolinsky, P. Boykin, and R. Figueiredo. Decentralized dynamic host configuration in wide-area overlays of virtual workstations. In *International Parallel and Distributed Processing Symposium*, pages 1–8, March 2007.

[14] B. Hubert. tc - linux advanced routing & traffic control. `http://lartc.org/`, June 2009.

[15] X. Jiang and D. Xu. Violin: Virtual internetworking on overlay. In *In Proc. of the 2nd Intl. Symp. on*

Parallel and Distributed Processing and Applications, pages 937–946, 2003.

[16] R. Jones. Netperf: A network performance monitoring tool. `http://www.netperf.org/netperf/NetperfPage.html`, 4 2009.

[17] D. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle. Ocala: an architecture for supporting legacy applications over overlays. In *NSDI'06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, pages 20–20, Berkeley, CA, USA, 2006. USENIX Association.

[18] M. Kallahalla, M. Uysal, R. Swaminathan, D. E. Lowell, M. Wray, T. Christian, N. Edwards, C. I. Dalton, and F. Gittler. SoftUDC: A software-based data center for utility computing. *Computer*, 37(11):38–46, 2004.

[19] K. Keahey and T. Freeman. Contextualization: Providing one-click virtual clusters. *IEEE International Conference on eScience*, pages 301–308, 2008.

[20] C. C. Keir, C. Clark, K. Fraser, S. H, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation*, pages 273–286, 2005.

[21] M. Krasnyansky. Universal tun/tap device driver. `http://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt`, March 2007.

[22] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP Journal*, 11(1), June 1997.

[23] M. Nelson, B.-H. Lim, and G. Hutchins. Fast transparent migration for virtual machines. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 25–25, Berkeley, CA, USA, 2005. USENIX Association.

[24] L. Peterson and D. Culler. Planet-lab. `http://www.planet-lab.org`, March 2007.

[25] J. Postel. Rfc 0925 - multi-lan address resolution, 1984.

[26] Qumranet. Kernel-based virtual machine for linux. `http://kvm.qumranet.com/kvmwiki`, March 2007.

[27] R. D. S. Alexander. *RFC 2132 DHCP Options and BOOTP Vendor Extensions*.

[28] Sun. gridengine. `http://gridengine.sunsource.net/`, March 2007.

[29] A. I. Sundararaj and P. A. Dinda. Towards virtual networks for virtual machine grid computing. In *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium*, pages 14–14, Berkeley, CA, USA, 2004. USENIX Association.

[30] M. Tsugawa and J. Fortes. A virtual network (vine) architecture for grid computing. *International Parallel and Distributed Processing Symposium*, 0:123, 2006.

[31] M. Tsugawa and J. Fortes. Characterizing user-level network virtualization: Performance, overheads and limits. In *IEEE International Conference on eScience*, pages 206–213, Dec. 2008.

[32] D. I. Wolinsky, Y. Liu, and R. Figueiredo. Towards a uniform self-configuring virtual private network for workstations and clusters in grid computing. In *VTDC '09: Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, pages 19–26, New York, NY, USA, 2009. ACM.

[33] J. Yonan. Openvpn. `http://openvpn.net/`, March 2007.