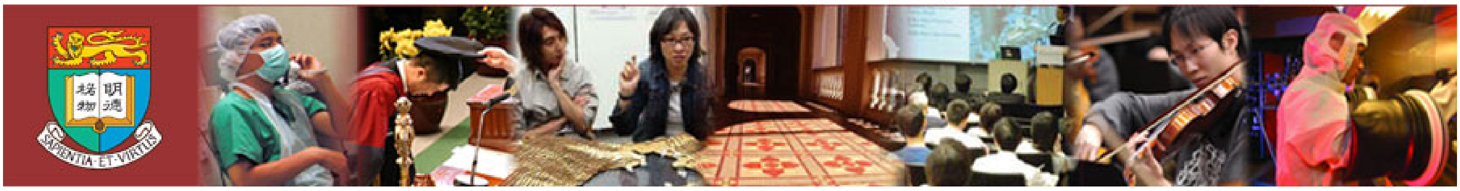


WAVNet: Wide-Area Network Virtualization Technique for Virtual Private Cloud

Zheming Xu, Sheng Di, Weida Zhang, Luwei Cheng, Cho-Li Wang
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
{zmxu, sdi, wdzhang, lwcheng, clwang}@cs.hku.hk

***Abstract*—A Virtual Private Cloud (VPC) is a secure collection of computing, storage and network resources spanning multiple sites over Wide Area Network (WAN). With VPC, computation and services are no longer restricted to a fixed site but can be relocated dynamically across geographical sites to improve manageability, performance and fault tolerance. We propose *WAVNet*, a layer 2 virtual private network (VPN) which supports virtual machine live migration over WAN to realize mobility of execution environment across multiple security domains. WAVNet adopts a UDP hole punching technique to achieve direct network connection between two Internet hosts without special router configuration. We evaluate our design in an emulated WAN with 64 hosts and also in a real WAN environment with 10 machines located at seven different sites across the Asia-Pacific region. The experimental results show that WAVNet not only achieves close-to-native host-to-host network bandwidth and latency, but also guarantees more effective VM live migration than existing solutions.**

I. Introduction

Cloud computing offers a new resource-sharing paradigm, namely Infrastructure as a Service (IaaS), which provides a level of abstraction and isolation over underlying physical resources. By multiplexing shared physical resources using virtualization technology, services encapsulated in virtual machines (VMs) can be delivered to end users on demand. With the increasing number of mobile applications deployed in various mobile terminals, the concept of cloud computing moves towards dynamic cloud service provisioning [1]. Cloud services are better off able to be autonomously migrated to those sites with more adequate resources, instead of competing for centralized resources at data centers. The new wave of such cloud paradigm has been witnessed by the recent cloud storage service providers such as Wuala [2] and Abacast [3], which leverage peer-to-peer (P2P) structure to provide online storage services or form content delivery network (CDN) by harnessing unused storage resource of desktop computers. Some projects (e.g. Clouds@home [4]) also aim to provide guaranteed computation power using Internet-connected volunteer resources. The expanding use of cloud services over Wide Area Network (WAN) makes the design of cloud platforms go towards a more flexible and distributed infrastructure with higher elasticity and mobility. Connecting volunteer resources together to serve as a unified computing infrastructure poses new challenges to network infrastructure as in reality, 60%∼80% hosts on the Internet are actually behind NAT/firewalls [5].

We propose WAVNet (short for Wide-Area Virtualized Network), a layer-2 network virtualization solution for dynamic construction of virtual private Cloud over a WAN environment. We leverage VM live migration technology [6] to realize such elastic Cloud Computing paradigm by dynamically connecting idle desktop computers behind the NAT/firewalls on the Internet. With WAVNet, each user can acquire a set of qualified hosts to run his/her tasks concurrently. New virtual machines can be instantiated locally and elastically scale-out by live VM migration to utilize remote computing resources as if they were in a secure and familiar local computing environment. WAVNet addresses the following problems:

- *Seamless network connection.* The wide deployment of NAT builds a barrier to dynamic construction of virtual private Cloud over a WAN environment as the hosts behind a NAT/firewall are only authorized to initiate outgoing traffic through a limited number of ports (UDP/TCP) but not authorized to receive incoming TCP or UDP traffic initiated by a foreign host. How to support transparent bi-directional network connectivity between any two computers (hosts or VMs) residing behind different NAT/firewalls and support live VM migration is an challenging issue.
- *Close-to-native transmission performance.* As the virtual network exists as an additional layer atop the native network, the overhead of processing redundant packet headers should be minimized.
- *Dynamic resource discovery protocol.* We target at users who want multiple non-dedicated computing resources to complete computation-intensive jobs, e.g. Bag-of-Task (BoT) applications and web based applications. We need a dynamic resource discovery protocol that is able to instantly locate idle hosts to run user tasks, while satisfying their specific computing requirements.

Various existing works have explored the area of virtual networking in pushing virtualization technology over WAN,

0190-3918/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPP.2011.90

yet none of them are suitable for the large-scale private cloud system. Traditional VPN [7], for example, adds a virtual IP layer on top of the physical IP layer, yet it requires a centralized server to forward network traffic, thus cannot scale to a large number of volunteer hosts. In [8], "socket remapping" was adopted as an optimized way of running MPI on virtual machines distributed on the Internet. Smartsocket [9] provides application libraries to traverse NAT/firewalls to achieve universal connectivity. However, these solutions are strictly bundled with specific applications and require re-implementation of every program that wants bi-directional connectivity. On the other hand, overlay networks [10], [11], [12], [13], [14] have been studied for years to achieve universal connectivity while appearing transparent to applications running on top. Whereas, these solutions either suffer from limited scalability [13], [15], or rely on special configurations on routers or gateways [11], introducing additional administrative burdens. In comparison, WAVNet adopts an additional light-weight virtual IP layer on top of IPv4 stack to build a virtual communication channel that can penetrate various NAT/firewalls, including Full Cone NAT, Restricted Cone NAT, and Restricted Port Cone NAT. Moreover, WAVNet also takes into account the locality issue to further optimize the resource grouping effect.

The rest of the paper is organized as follows. In Section II, we present our core design on network virtualization, a.k.a. WAVNet. We mainly consider our contributions on the performance-oriented design of WAVNet architecture (Section II.A) in practice and a set of carefully devised strategies for improving the virtual network efficiency (Section II.B $\sim$ II.D). We describe the experimental configurations and analyze the performance statistics in Section III. The related works are discussed in Section IV. We conclude our work and point out future directions in Section V.

II. WAVNet

WAVNet provides a link-layer virtual networking infrastructure for constructing a private cloud over a WAN environment. In order to minimize the cost/overhead of maintaining idle connections between such NATed hosts, we adopt a two-layer architecture to organize all the hosts.

As shown in Figure 1, any physical host (e.g. desktop PC with a private IP address behind NAT) could join the WAVNet by sending a joining message to at least one *rendezvous server* with a public IP address. The rendezvous server will record the new host's NAT server IP address and port number. The connection between the host and its rendezvous server needs to be maintained since the rendezvous server has to notify new connection requests to this host from time to time. All of rendezvous servers are organized by Content Addressable Network (CAN) [16][1], and each of them serves as a self-managed peer node/proxy with resource lookup services and network distance detecting services. The rendezvous server could be cluster's frontend gatekeeper host with a public IP address. If the joining host has no knowledge about its NAT server IP address, it could connect to some designated rendezvous servers on Internet for joining WAVNet. Each rendezvous server periodically communicates with others over CAN overlay to share the state information of resource hosts (i.e. desktop hosts).

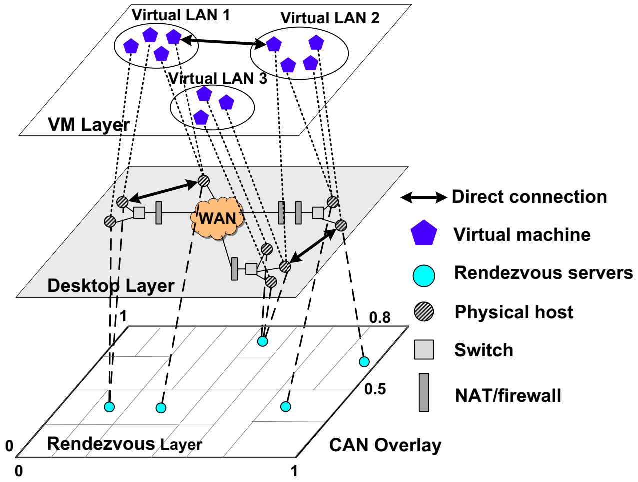


Figure 1. Conceptual view of WAVNet

Any user could raise a resource query from his/her own desktop PC through its rendezvous server.Whenever some resource hosts are found by routing query message among the rendezvous servers, direct connections (arrows in Figure 1) between involved desktop computers or VMs are established via WAVNet's connection setup procedures. This process connects the resources as if they were connected to the same Ethernet switch. By explicitly bridging to the hosts' WAVNet interfaces, VMs are also plugged into the same link-layer virtual network (virtual LANs in Figure 1). In this way, provisioning of a virtual private cloud can be carried out by either requesting instantiation of VM on remote resources, or migrating customized VMs to remote idle hosts. These VMs can be accessed by unmodified remote control interfaces such as *remote shell*.

In a dynamic virtual network environment, resources may join and leave. This requires the underlying network infrastructure to adapt to a frequently-changing working environment. By leveraging WAVNet as the virtual networking infrastructure, VMs over the same virtual LAN could be migrated freely across different security domains without interrupting the task execution states and network connection.

A. System Modules

Figure 2 illustrates the system modules of WAVNet. Communication between hosts is categorized into two types: (1) overlay messages between itself and rendezvous servers

[1] Due to the P2P structure of the rendezvous layer, we call a rendezvous server *node* and a desktop computer *host*, respectively, in following text.

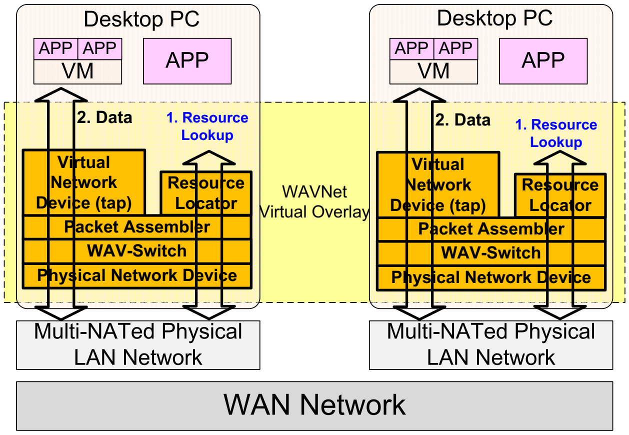


Figure 2. Overview of WAVNet architecture

and (2) data transmitted over direct host-to-host connection. The former type of communication usually takes place during host join/leave, resource lookup and so on, while the latter happens when network applications perform data transmission. Application data is captured by the user-level Virtual Network Device (*tap*), and handed to Packet Assembler (PA). PA is used to categorize communication packets and encapsulate them with proper identifiers. The Wide-Area Virtual Switch (WAV-Switch) functions like an Ethernet switch: it inspects the hardware address of communication packets and determines the connection over which the packets will be sent. The difference of WAV-Switch from an ordinary hardware switch is that WAV-Switch works for WAN network while a hardware switch only functions in LAN. After the destination connection is settled down, the packets are multiplexed over the underlying physical network device. Resource lookup module is in charge of the overlay message communication and the basic connection maintenance (to be discussed in next section) with its rendezvous server. Although a DHT overlay is used for resource lookup, the actual data transmission between any desktop hosts after the connection is established does not involve the DHT overlay. Such a design avoids the additional DHT-layer header and processing overhead during data transmission.

B. Direct Host-to-host Connection

Direct host-to-host connection contains two key operations: connection establishment and connection maintenance. As described previously, prevalent NAT deployments prevent Internet hosts from establishing bi-directional connections. Since NAT is not standardized, different Internet service providers could adopt different policies of port mapping. We leverage STUN protocol to provide a way of querying the public IP address and port information of a NATed host since STUN can detect different types of NAT: Full Cone NAT, Restricted Cone NAT, Restricted Port Cone NAT or Symmetric NAT. With STUN, WAVNet driver could determine if the host is suitable for UDP hole punching or not. UDP hole punching enables two hosts to set up a direct peer-to-peer UDP session with the help of a well-known rendezvous server, even if the hosts are both behind NATs. In WAVNet, all host-to-host connections are built based on the combination of STUN protocol and UDP hole punching techniques. Such techniques could traverse most of real-life NATs [14].

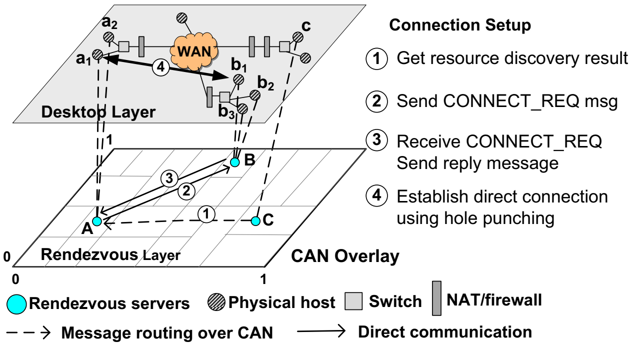


Figure 3. Host-to-host connection setup

Different from traditional UDP hole punching, the UDP hole punching in WAVNet is integrated as part of the resource query layer (CAN overlay) designed for users to locate most suitable resources in the virtual private cloud. Resource queries are routed through rendezvous servers, and the result is returned to the requester. The host's connection information, including its rendezvous server's IP address and a 2-tuple {NAT server IP: NAT server port} detected when penetrating its NAT, is encapsulated in the query result. We give an example with 2-dimensional CAN space in Figure 3 to illustrate the connection setup procedure. In this example, suppose node C maintains the resource information about host b_1 because b_1's state (a multi-dimensional vector) is right overlapped with node C's zone range. First, the requester host a_1 makes use of its rendezvous server node A to get the host b_1's information, including the resource state and the connection information, through CAN overlay (step 1). Then, two rendezvous servers (A and B) will communicate with each other to notify the two hosts a_1 and b_1 the mutual connection information (step 2 and step 3). Finally, host a_1 and b_1 build direction connection using hole punching (step 4).

UDP hole punching adopted by WAVNet avoids any special configurations on routers or gateways. This makes the whole process of connection setup quite easy. Suppose user wants host a_1 to connect b_1, he/she first downloads the WAVNet driver, which is already configured with well-known rendezvous server(s). WAVNet will automatically communicate with the rendezvous server(s) to help the user find b_1 according to user's description. After a_1 gets the public information of b_1, the WAVNet driver will automatically connect the hosts through UDP hole punching.

After the connection setup, the two hosts are connected

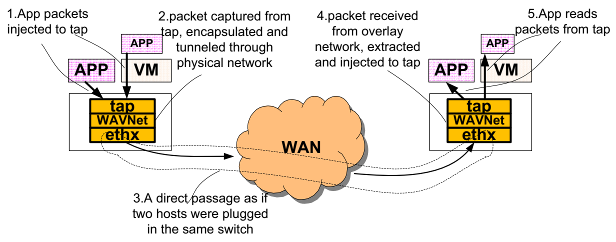

1.App packets injected to tap
2.packet captured from tap, encapsulated and tunneled through physical network
4.packet received from overlay network, extracted and injected to tap
5.App reads packets from tap
APP
APP
VM
tap
WAVNet
ethx
APP
APP
VM
tap
WAVNet
ethx
WAN
3.A direct passage as if two hosts were plugged in the same switch

Figure 4. Direct host-to-host tunneling in WAVNet

as if to an Ethernet switch. Therefore, protocols such as DHCP can be applied without any modification. Moreover, application data transmission between the connected hosts will not go through the CAN overlay, which considerably saves packet header overhead as well as protocol processing time. Detailed traveling path for each application packet is shown in Figure 4: once host-to-host direct connection is established, applications communicate through the virtual network device (*tap*). Packets injected to the virtual network device on the sending end will be captured and tunneled through the physical network, and transmitted to the receiving end of the connection, where the original packet is extracted and injected to the virtual network device. The communication does not rely on intermediate CAN-overlay nodes for routing.

One remaining issue is that connections between NATed hosts must be deliberately kept alive, otherwise the connection will be lost because NAT can only maintain the connection state for a limited period of time. Thus, periodical exchange of messages over established connections must be scheduled such that NAT can re-count the timeout of the existing connections. However, periodical exchange of ping messages incurs bandwidth and processing overhead [17]. Therefore, we provide a lightweight CONNECT_PULSE message, which minimizes the overhead by containing only a header with two bytes. During the exchange of such messages, NATs along the connection path get notified that there is still traffic going through the opened port, thus keeping the port always active.

C. Live VM Migration over WAN

Current Virtual Machine Monitors (VMM) support live migration within LAN by adopting the bridging mode for VM networking since NAT and Router modes do not maintain network connections. In the bridging mode implemented by Xen [18], virtual machines have front-end network drivers that interface with users and back-end drivers in the driver domain (Domain-0). To make virtual machines stay on the same link layer as other hosts in LAN, a software bridge is created, with virtual machines' back-end drivers and physical host's external network devices as software ports. The key to supporting seamless live migration lies in the link layer network. When live migration finishes and the virtual machine is brought up on the destination host, the VMM will inject an unsolicited ARP broadcast into the software bridge on behalf of the virtual machine. All physical hosts and virtual machines in the same LAN will receive the ARP frame and update the location of the migrated virtual machine in the local cache. Since applications are usually based on IP protocol, the update of link-layer address does not disrupt the consistency of connection. The ongoing IP packets will be sent to the new location of the virtual machine. Nevertheless, such a seamless live migration cannot be applied to WAN because of the connectivity problem caused by NATs.

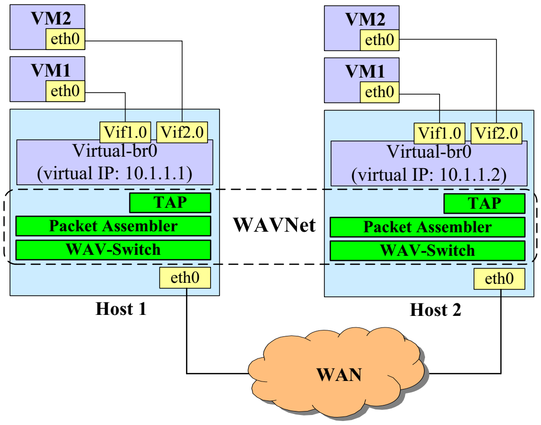


Figure 5. WAN based Live migration support by WAVNet

Figure 5 demonstrates the approach adopted in WAVNet to enable seamless VM live migration over WAN. Suppose host 1 and host 2 are connected across WAN according to the procedures shown in previous section. Therefore link-layer connection is established between these two hosts. A dedicated virtual network bridge is created with the *tap* device as the external port. Link layer frames injected by virtual machines and VMM will be extracted by WAVNet and tunneled to the other end. When VM live migration is performed, the ARP broadcast will be forwarded by WAVNet to all connected hosts. Existing open connections to the migrated virtual machine will not be disrupted and ongoing data stream will not be confused with the location change of the virtual machine. This is fundamentally different from the solutions that are built upon layer-3 overlays and resorted to DHT updates by broadcasting the information. Such approach may lead to much longer downtime time of the migrating virtual machine perceived by other hosts.

D. Host Selection for Virtual Cluster

Using the CAN protocol, each host could locate a set of other resource hosts quickly, by taking multiple attributes into account , such as available CPU and memory. For some Bag-of-Task applications which are bandwidth or latency sensitive (e.g. MPI tasks and FTP/SCP services), the group of hosts selected need to communicate with each other with high connection quality. By using a *distance locator* deployed in each rendezvous server, we could dynamically locate a set of mutually near hosts in WAVNet to construct a virtual cluster according to users' demand (e.g. number of hosts they need).

By analyzing the mutual connection latency among the WAN hosts, we devise an approximation optimal algorithm performed at *distance locators* for host selection. We model the grouping problem as a combinatorial optimization problem: given N candidate hosts, we construct an $N \times N$ matrix, whose elements are the mutual network latency. The objective is to find a group of k hosts such that the average network latency among these hosts is minimized (i.e. Formula (1), where Π refers to the candidate resource set).

$$\overline{L}(\Pi) = \frac{\sum_{x,y \in \Pi} latency(x,y)}{C^{2}_{|\Pi|}} \tag{1}$$

Although the grouping problem mentioned above is not NP-complete because we could verify all the solutions (a.k.a. brute-force method) in C^{k}_{N} steps, the time complexity ($O(N^{k})$) looking for the optimal solution is still too high to be tolerable in practice. Hence, we design a novel approximation algorithm of grouping hosts in vicinity to optimize the constructed virtual cluster's execution efficiency.

Our model assume the WAN network latency conforms to symmetric property and transitive property with high probability, as illustrated in Formula (2) and Formula (3), where $\triangleright$ means that the former host pings the latter host with low latency. Such properties were also observed on hosts connected in Planetlab [19].

$$Host\ A\ \triangleright\ Host\ B\ \Rightarrow\ Host\ B\ \triangleright\ Host\ A \tag{2}$$

$$HostA \triangleright HostB, HostB \triangleright HostC \Rightarrow HostA \triangleright HostC \tag{3}$$

Our algorithm is performed on each *distance locator* co-located with the rendezvous server. The algorithm consists of two parts. The first part is to maintain a latency matrix by pefiodically communicating with neighbors. Each row in this matrix is always sorted in an increasing order. Upon receiving a request, only the second part (a.k.a. grouping algorithm) would be triggered to generate the group of Internet-connected hosts with optimized mutual communication status on WAN. Obviously, the response time of any request is only related to the complexity of the grouping algorithm (i.e. second part). Suppose there are N candidate hosts aggregated on the rendevous server and k hosts or VMs are to be selected for user task's consumption. Based on the latency matrix with sorted connection elements on each row, the grouping algorithm will group the first k+1 elements (namely k+1-group) at each row, and create k different combinations using any k elements (namely k-group) from the k+1 elements. And then, the algorithm will check all the $k \cdot N$ candidate solutions and filter those with at least one unreasonable or over-large connection. Finally, the best k-group from the remaining choices with the minimal average latency will be selected. It is easy to see that the time complexity of the grouping algorithm is $O(N \cdot \mathrm{k})$, which is far less than that of brute-force method ($O(N^{k})$).

III. Experiments and Evaluations

We evaluate the performance of WAVNet under two different environments: (1) a real WAN environment with machines located at seven different geographical sites as shown in Table I, each machine running Xen 3.1 and Linux 2.6 operating system. The 3rd column shows the ping latency from HKU to each site. It should be noted that KVM virtual machines are used in two sites to validate the generality of our solution. One rendezvous server with public IP addresses is configured at Hong Kong. (2) an emulated WAN environment with up to 64 machines connected by four fast Ethernet switches. Eight machines are equipped with 32GB memory and two quad-core 2.6GHz E5540 Xeon CPU. The rest are commodity PCs, each with one 2.2GHz Pentium 4 CPU and 2GB memory. The NAT/firewalls on gateway PCs is configured by adding rules in *iptables nat* table. We use Linux *traffic controller* (tc) to shape network bandwidth for simulating different network conditions. We show that WAVNet is not only able to support VM live migration over WAN effectively, but also deliver advantageous performance over IPOP [10] in various conditions. We also evaluate the performance of the proposed locality-sensitive grouping strategy over Planetlab [19].

Table I
HOST CONFIGURATION IN A REAL WAN ENVIRONMENT

Sites	**Machine Info.**	**Lat. (ms)**
Providence University (PU), Taiwan	Intel Core 2 Quad Q6600 2.40GHz (4085MB)	30.2
Academia Sinica (Sinica), Taiwan	Intel Xeon E5520 2.27GHz (KVM with 2 cores, 8183MB)	24.8
Advanced Industrial Sci. and Tech. (AIST), Japan	Intel Core 2 Duo E6300 1.86GHz (3191MB)	75.8
San Diego Supercomputer Center (SDSC), USA	Intel Xeon 3.20GHz (KVM with 4 cores, 16383MB)	271.2
The University of Hong Kong (HKU), HK	Intel Core 2 Duo T7250 3.20GHz (1526MB) and Intel Pentium 4 2.80GHz (1526MB)	0.5
Home PC connected by public network (OffCam), HK	Intel Pentium 4 2.80GHz (1279MB)	4.4
Shenzhen Inst. of Advanced Tech. (SIAT), China	Intel Pentium 4 2.80GHz (1279MB)	74.2

A. Link Throughput and Latency

We first compare latency measurements of WAVNet with IPOP in the real WAN environment, with respect to the physical network. The latency measurement was done by performing ICMP echo test, each lasting 10 minutes to ensure the accuracy of the evaluation. The result is summarized in Table II. Due to the long distance connection, the packet handling overheads were amortized by the long network latency. So both WAVNet and IPOP achieve performance equally well and are close to the physical network.

We use *ttcp* to measure the bandwidth of network connections under IPOP and WAVNet over WAN (HKU-SIAT). In Figure 6, it is observed that both WAVNet and IPOP

Table II
Network Latency Test by ICMP Request/Response

Sites	Mean Round-Trip Time (msec)		
	Physical	WAVNet	IPOP
HKU-SIAT	74.244	74.207	74.596
HKU-PU	30.233	30.753	31.187
SIAT-PU	219.427	219.783	220.533

achieves 57% to 85% of the physical network bandwidth, yet in almost all cases WAVNet outperforms IPOP. This reflects that WAVNet imposes less overhead in handling the packets than IPOP.

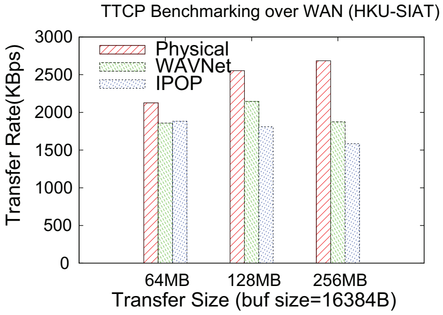


Figure 6. TTCP bandwidth benchmark

We further evaluate the bandwidth performance of end-to-end connection built on WAVNet virtual network, under the emulated WAN environment. Netperf TCP STREAM test is performed to generate network traffic, with duration of 360 seconds. The average statistic of 10 tests leads to the final result. As shown in Figure 7, in all cases, WAVNet has near-to-native performance. This confirms that the processing overhead of WAVNet is small. We also evaluate the performance of IPOP under the same network conditions. When the network is highly congested (e.g. WAN bandwidth is small), IPOP performs slightly worse than WAVNet. When the underlying network is less congested, IPOP shows a deficient performance, which is less than 20% of the native performance. This reveals that WAVNet can better utilize the available physical bandwidth, particularly when the network capacity of WAN/MAN is large.

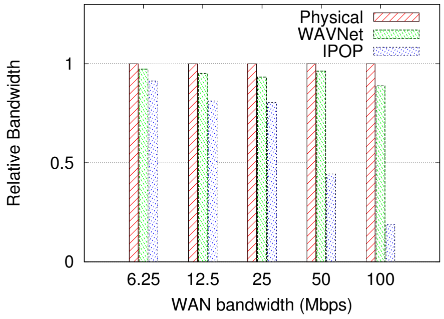


Figure 7. Bandwidth utilization under different network conditions

B. Scalability of Virtual Networking

WAVNet requires periodical exchange of messages over established connections to maintain the connection state, which may affect the scalability of virtual networking. For instance, in the case of 64-host cluster, every node has to establish 63 direct host-to-host connections with other hosts. In the experiment, we setup virtual clusters connected through WAVNet with 8, 16, 24, 32, 48, 64 hosts. We select one node in the virtual cluster and use Netperf to measure the host-to-host network bandwidth from this node to the rest of nodes and calculate the average bandwidth. We analyze the overheads of host-to-host connections in maintaining such virtual cluster in WAVNet by measuring the Netperf performance under different virtual cluster sizes.

We set the period of message exchange for keeping connections alive to be 5 seconds, which is short enough in comparison with NAT's timeout (usually a couple of minutes). Figure 8 shows that in a virtual cluster consisting of 64 hosts, bandwidth performance for each host is not degraded compared with those with smaller number of hosts (e.g. 8 hosts). On the other hand, IPOP suffers a notably degraded performance as the number of hosts increases, due to the extra overheads occurred in data transmission over multiple intermediate routing hosts based on its rigid routing algorithm. With increasing number of WAN hosts, the number of intermediate routing hosts is expected to increase under IPOP, which leads to a decreasing performance. The physical topology would also severely affect the performance in IPOP. WAVNet bypasses overlay routing while performing data transmission. Netperf performance under WAVNet is relatively more consistent even the cluster size increases.

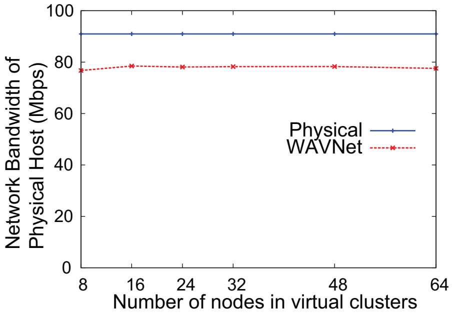


Figure 8. The Netperf performance while scaling virtual cluster size

C. VM Live Migration over WAN

We evaluate live VM migration over the emulated WAN and the real WAN environment based on the Netperf TCP_STREAM tests. We demonstrate that WAVNet could perform seamless VM live migration with persistent network connectivity. We use the Netperf TCP_STREAM tests to poll the TCP transmission performance, which is reported every 500ms. The virtual machine is installed with CentOS. VM migration is triggered manually, sometime after polling process starts.

We first report the network bandwidth of the VM during live migration under the emulated WAN environment. Each

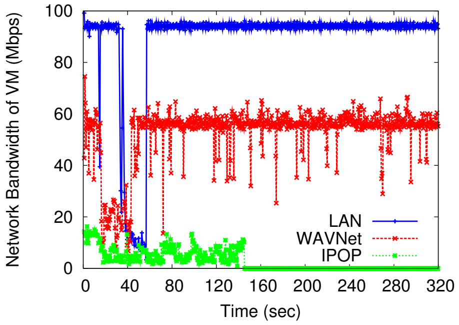


Figure 9. VM Network Bandwidth Test During Live Migration

Table III
HTTP CONNECTION TIME BEFORE/AFTER VM MIGRATION

Client and VM Location	Ping Lat. (msec)	Conn. Time (msec)		
		Min	Mean	Max
Sinica to VM@SIAT (before migr.)	100.3	99	107	148
Sinica to VM@HKU$_2$ (after migr.)	24.8	25	33	67
HKU$_1$ to VM@SIAT (before migr.)	74.2	76	80	90
HKU$_1$ to VM@HKU$_2$ (after migr.)	0.5	0	7	16

Table IV
HTTP THROUGHPUT BEFORE/AFTER VM MIGRATION

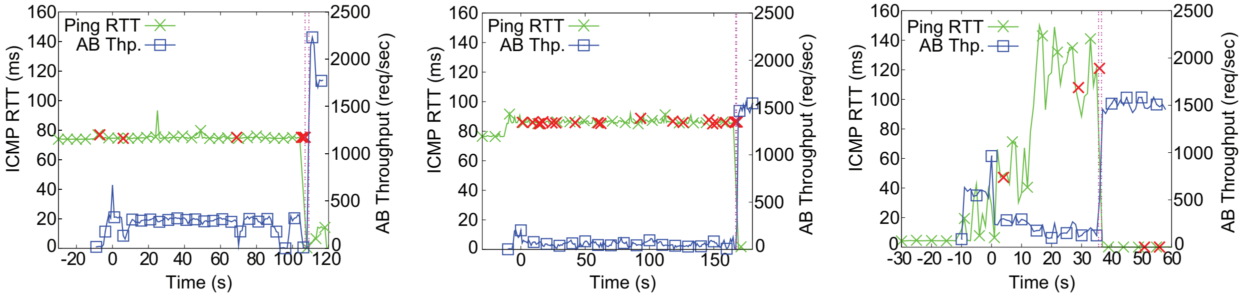
Client and VM Location	WAVNet bw (Mbps)	AB Thp. (# req. /sec)		
		1K	8K	64K
Sinica to VM@SIAT (before migr.)	18.05	432.9	215.1	45.7
Sinica to VM@HKU$_2$ (after migr.)	21.69	583.3	332.3	53.9
HKU$_1$ to VM@SIAT (before migr.)	18.6	473.1	288.9	56.9
HKU$_1$ to VM@HKU$_2$ (after migr.)	79.15	775.5	461.8	128.2

VM is configured with a memory size of 256MB. Figure 9 compares the measured performance under IPOP, WAVNet, and native network (denoted as "LAN"). VM migration takes around 20 seconds under LAN and Netperf reports about 95% of native bandwidth. IPOP achieves less than 10% of the native bandwidth, while VM migration takes around 130 seconds. After the VM is migrated, the Netperf test is even stalled in IPOP, because IPOP is not aware of the move of the VM, and continues routing the packets to the source host. WAVNet can achieve around 60% of the native network bandwidth, meanwhile the migration takes less than 30 seconds and the Netperf session could continue after the live migration finishes. We also use *tcpdump* to capture the link-layer ARP broadcast that is dispatched when live migration finishes. tcpdump actively listens to the network interface for ARP frames that pass by. In LAN, when live migration of VM is finished, an ARP frame is captured by tcpdump and the content is printed. When we use WAVNet and let tcpdump listen to the tap device, a similar ARP frame is also captured, reflecting that WAVNet could tunnel the link-layer frames in supporting VM migration over the emulated WAN.

To evaluate the effectiveness and performance of VM live migration across a real WAN environment, we use rendezvous server layer to establish the connection among hosts from different geographical sites, including Hong Kong, mainland China, Taiwan, Japan and United States. It should be noted that the rendezvous server does not involve in the host-to-host communication after the connection is established, as explained in Section II.

Firstly, we evaluate how VM migration can improve the access locality by migrating http server for better request throughput and shorter http connection time. We set up an http server on a VM with 128MB memory located in SIAT, one http client in HKU$_1$ and another http client in Sinica. The VM in SIAT will be migrated to another host in HKU (denoted as HKU$_2$). In client side we use *ApacheBench* (AB), a web site stress test benchmark, to measure the request connection time and throughput for the underlying WAVNet network before/after VM migration. Table III shows the http connection time results obtained from HKU$_1$ client and Sinica client, and Table IV shows http request throughput with different requested file sizes. In Sinica client, the network latency to VM in SIAT is around 100ms with Netperf TCP throughput of 18.05Mbits/sec, while after VM is migrated to HKU$_2$ over WAVNet, network latency to VM is 24.8ms and the Netperf TCP throughput reports 21.69Mbits/sec. Similarly in HKU$_1$ client, after VM migration the network latency to VM improves from 74.2ms to 0.5ms, and Netperf TCP throughput improves from 18.6Mbits/sec to 79.15Mbits/sec. Since the underlying network condition is much better after VM migration, the http connection time and http throughput are both improved. This reflects that better user experience is achieved, as http server can be migrated to a nearby host to provide more responsive service.

We further analyze the VM down time, ICMP packet loss, and provide a micro-view on ping latency and HTTP throughput during VM live migration. The purpose is to report the service quality and availability during the period of VM live migration. As shown in Figure 10, we individually migrate the VM from OffCam, AIST and SIAT to a host in HKU. During VM live migration, we run ApacheBench on another host in HKU to request for a 1KB file from the VM, with concurrency set at 50 for illustration purpose. Meanwhile, we use ping to measure the network latency and reflect packet loss during VM live migration. In Figure 10 (a)-(c), time zero represents the moment that VM live migration is triggered. The duration of VM down time is shown as the short interval between two vertical dashed lines. For example in Figure 10(a), the ping test is started 30 seconds before VM live migration is triggered and the client starts sending http requests 10 seconds before that. It can be seen that after we start sending http requests, ping test reports higher RTT due to heavy network traffic generated by http requests. When VM migration is triggered, HTTP throughput drops from 600 requests/sec to nearly 300 requests/sec, meanwhile ping test also suffers packet loss. Once the VM migration is finished and relocated in HKU which is very near to the testing clients, significant improvements can be observed that the throughput increases to over 1500 requests/sec and ping latency decreases to less than 15ms. Similar phenomena can be found in Figure 10(b) and



(a) AIST-HKU, VM down time is 2.1s.

(b) SIAT-HKU, VM down time is 1.0s.

(c) OffCam-HKU, VM down time is 0.6s.

Figure 10. ICMP RTT and HTTP throughput during VM live migration (× represents ICMP packet loss)

Table V
TIME OF VM LIVE MIGRATION AMONG DIFFERENT SITES

Sites	RTT (ms)	WAVNet bw (Mbps)	Time taken (s)	
			128M	512M
OffCam-HKU	4.4	86.39	16	120
Sinica-HKU	24.8	42.93	92.5	202.5
AIST-HKU	75.8	55.1	107.5	208
SIAT-HKU	74.2	18.6	130	377.5
SDSC-HKU	217.2	27.17	310.5	1023

(c). As OffCam-HKU has a much higher network bandwidth, the heavy data traffic passing through the software bridge might interfere ping tests and result in higher jitter.

We also evaluate the migration time of VM with different memory sizes under different network conditions in Table V. We test the VM memory size of 128MB and 512MB, and individually migrate the VM from OffCam, Sinica, AIST, SIAT and SDSC to HKU. The testing results show that under the same physical network, bigger VM memory size results in longer migration time as more data is needed to transmit. Also, the network with low latency and high bandwidth can benefit VM migration time in various aspects. For instance, the VM migration time of AIST-HKU is three to five times faster than that of SDSC-HKU. We also found that under the same network condition, VM migration time is not always proportional to the VM memory size, as Xen adopts pre-copy strategy to transmit dirty memory pages in several rounds[6]. The first round Xen transmits all the pages to destination host. However, as the VM at the source node is still alive and might continuously update its memory pages, in each round afterwards, Xen transmits the pages updated in previous round. The larger the network latency is, the longer each round takes. Thus, more dirty pages are likely to be generated, and more data volume will be migrated in each round.

We evaluate the performance benefit that VM live migration brings to parallel applications over WAN. We implement the MPI program of heat distribution problem [20], and set up the experiment environment as follows: four virtual machines are connected through WAVNet to run MPI heat distribution test, with three of them located in HKU and one located in SIAT. We then measure different problem sizes of 64×64, 128×128 and 256×256 respectively, where $m \times m$ refers to a square with m^2 uniformly distributed sensor points. Figure 11 shows that without VM migration, the MPI tasks for problem size 64×64, 128×128, 256×256 last for 397s, 1214s and 3798s apiece. Comparably in the second test, we migrate the VM in SIAT to one host in HKU after the program starts. Results show that the MPI tasks with VM migration last for 121s, 179s and 365s respectively, which are 30.5%, 14.7% and 4.7% of the time without VM migration. This is because the VMs are closer to each other after the VM migration, while without VM migration, the communication between SIAT and HKU is the bottleneck. Lastly, the execution of MPI application is not disrupted during the migration process, which also proves WAVNet's ability to support seamless VM live migration with persistent network connectivity.

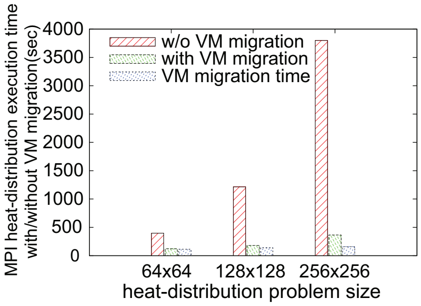
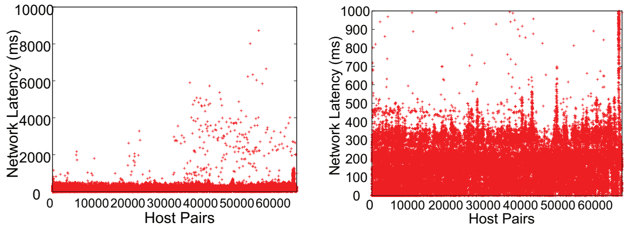


Figure 11. MPICH heat distribution test

D. Locality-sensitive Grouping Strategy on Planetlab

We evaluate our locality-sensitive grouping strategy over Planetlab among 400 randomly selected hosts around the world, using NAS MPI parallel benchmark [21]. Among the 400 hosts, there should be P_{400}^{2}=159600 bidirectional connections. Based on the symmetrical property of network latency (Formula (2)), we used about half number of the connections (80000 connections) to observe the network status. Figure 12 (a) and (b) show the network latency distributions within 10 seconds and 1 second respectively.

Figure 13 shows the average latency (i.e. $\overline{L}(\Pi)$ calculated by our grouping algorithm (Formula (1)), as well as the lowerbound and upperbound of the latency range, where the



(a) Network latency within 10 sec (b) Network latency within 1 sec

Figure 12. Network latency reported on Planetlab (400 hosts)

number of hosts involved are 2∼75. In our test, when the number of hosts of virtual cluster is individually set to be 8, 16, 32 and 64, the average latency is only 1.3ms, 15.4ms, 26.1ms and 54.1ms respectively, with the upperbound latency be 1.9, 25.4, 44.8 and 67.3. The results confirm the effectiveness of our designed locality-sensitive strategy especially compared to the original distribution (Figure 12).

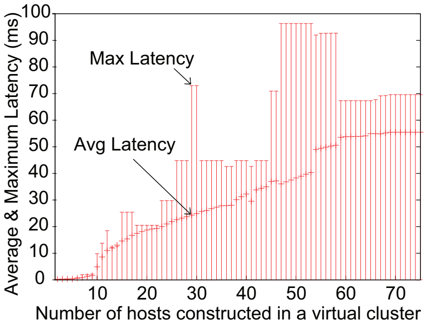
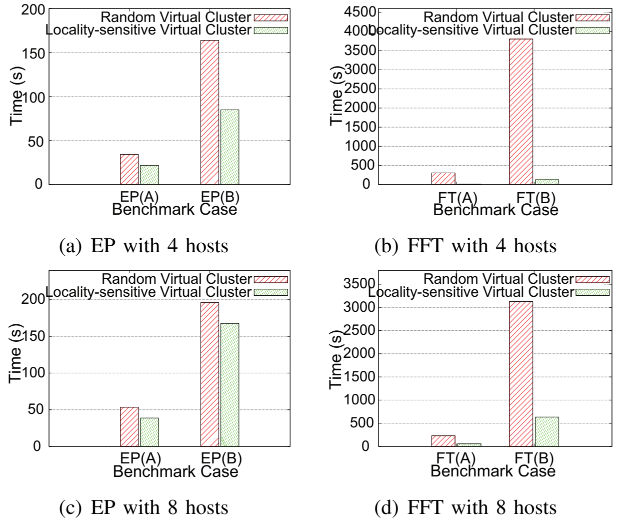


Figure 13. Average and Maximum Latency within Virtual Cluster

We choose an embarrassing parallel (EP) case and a non-embarrassing parallel (NP) case in NAS benchmark library and test them on a small virtual cluster. The NP case is solved using Fast Fourier transform (FFT) program. Figure 14 shows the experimental results using 4 hosts and 8 hosts. In the comparative case, 4 or 8 hosts are randomly chosen from 64 hosts pre-selected by our locality-sensitive grouping method, in order to guarantee that the selected hosts still have reasonable inter-connectability between each other. As seen from the two figures, the locality-sensitive method could effectively improve the communication quality of virtual cluster constructed on WAVNet, particularly for non-embarrassing parallel (NP) case because FFT highly relies on the inter-host communication.



(a) EP with 4 hosts (b) FFT with 4 hosts

(c) EP with 8 hosts (d) FFT with 8 hosts

Figure 14. Locality-sensitive Method vs. Random Selection Method

IV. Related Work

Overlay network [10], [11], [12], [13], [14], [22] is the main track of existing network virtulization research, which aims to achieve universal connectivity meanwhile appear transparent to the applications running above. VOILIN [13], [15] proposes "vSwitch" and "vRouter" as virtual networking infrastructure to simulate physical networking devices, but all network traffic must go through the virtual network devices, which limits its scalability. VNET [12], [23] is a layer-2 solution for tunneling Ethernet frames, yet it requires fixed public servers to act as proxies to tunnel network traffic for end hosts and cannot group hosts for users to optimize the intro-group communication. ViNe [11] provides a dynamic routing infrastructure to allow connectivity and isolate virtual clusters over multiple sites, yet requires routers and gateways to be specially configured, which potentially involves additional administrative effort every time when new sites join. Some other solutions, like CloudNet [24], make use of *VPLS* [25] technology and provide bi-directional network connectivity among Internet hosts, but alo require every router to be specially configured to form a giant "Internet switch", introducing non-ignorable burdens to users.

The most similar work to WAVNet is perhaps IPOP [17], which also makes use of NAT hole punching techniques and a user-level virtual network device to build overlay networks. Although this solution is self-configured and scalable to certain extent, it does not address the following issues: 1) Processing overhead of each packet through the additional P2P routing layer severely degrades the network performance. Such performance might offset the advantages that virtual networks bring, making it more difficult to be accepted. 2) It confines the number of direct host-to-host connections each peer is able to maintain. While in reality, users tend to run parallel programs that rely on multiple high-performing direct connections for inter-process communications. 3) Whenever virtual machines migrate, the IPOP program needs to be killed and restarted at the destination (for interface/hybrid mode) or all other relevant peers in the P2P overlay needs to be informed of the updated location of the migrating VM (for router mode), which interrupts all connections that the VM maintains prior to migration. This is undesirable in situations when VMs are likely to re-locate during the execution of parallel tasks, for load-balance or fault-tolerance purposes. 4) In terms of resource discovery, P2P overlay of IPOP is not aware of physical resource availability of peers except for aimlessly checking the existence of IP addresses, which is neither meaningful nor efficient from a user's point of view.

V. Conclusion

In this paper, we present a performance-oriented network virtualization model, WAVNet, which can well adapt to dynamic provisioning of IaaS over the large-scale wide-area network. WAVNet provides a link-layer virtual network that tunnels application packets for any Internet-connected hosts even behind NAT/firewalls. Direct host-to-host connection among resources discovered over rendezvous layer (CAN overlay) allows users to utilize the available physical bandwidth with minimal cost. On top of WAVNet, users can also build their own virtual clusters that could expand or shrink in number of available resources. Seamless WAN-based VM live migration, a key technology supporting fault tolerance and load balance for large-scale cloud system, is implemented in our WAVNet. Our experiments show that WAVNet delivers advantageous performance over the previous solutions, not only about the VM live migration but also on various practical applications. Experiments reveal that parallel computation (such as MPI programs) can not only be executed in a local-area network, but also be transparently/conveniently carried out over wide-area network efficiently using our designed WAVNet.

Acknowledgments

This research is supported by a Hong Kong RGC grant HKU 7179/09E and a HKU Basic Research grant (Grant No. 10401460), and also in part by a Hong Kong UGC Special Equipment Grant (SEG HKU09). Special thanks also to SIAT, AIST, SDSC, Sinica and PU, for their kind help in providing machines.

References

[1] R. K. K. Ma, K. T. Lam, C.-L. Wang, and C. Zhang, "A stack-on-demand execution model for elastic computing," in *Proc. 39th Int. Conf. Parallel Processing*, 2010, pp. 208–217.

[2] Wuala: http://www.wuala.com/.

[3] Abacast: http://www.abacast.com/.

[4] Clouds@home: http://clouds.gforge.inria.fr.

[5] Y. Huang, T. Z. J. Fu, D. M. Chiu, J. C. S. Lui, and C. Huang, "Challenges, design and analysis of a large-scale p2p-vod system," in *Proc. 2003 Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2008, pp. 375–388.

[6] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proc. 2th USENIX Conf. Networked Systems Design and Implementation*, 2005, pp. 273–286.

[7] OpenVPN: http://openvpn.net/.

[8] Z. Pan, X. Ren, R. Eigenmann, and D. Xu, "Executing MPI programs on virtual machines in an internet sharing system," in *Proc. 20th Int. Parallel and Distributed Processing Symposium*, 2006, pp. 101–110.

[9] J. Maassen and H. E. Bal, "Smartsockets: solving the connectivity problems in grid computing," in *Proc. 16th Int. Symposium on High Performance Distributed Computing*, 2007.

[10] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "IP over P2P: Enabling self-configuring virtual ip networks for grid computing," in *Proc. 20th Int. Parallel and Distributed Processing Symposium*, 2006.

[11] M. Tsugawa and J. Fortes, "A virtual network (ViNe) architecture for grid computing," *Proc. 20th Int. Parallel and Distributed Processing Symposium*, pp. 10–19, 2006.

[12] A. I. Sundararaj and P. A. Dinda, "Towards virtual networks for virtual machine grid computing," in *Proc. 3rd Conf. Virtual Machine Research And Technology Symposium*, 2004.

[13] X. Jiang and D. Xu, "VIOLIN: Virtual internetworking on overlay infrastructure," in *Parallel and Distributed Processing and Applications*, December 2004.

[14] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *Proc. USENIX Annual Technical Conf.*, 2005, pp. 179–192.

[15] 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. 3rd IEEE Int. Conf. Autonomic Computing* , 2006, pp. 5–14.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. 2001 Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications*, vol. 31, no. 4, October 2001, pp. 161–172.

[17] A. Ganguly, A. Agrawal, P. O. Boykin, and R. Figueiredo, "WOW: Self-organizing wide area overlay networks of virtual workstations," in *Proc. 15th Int. Symposium on High Performance Distributed Computing*, vol. 5, 2006, pp. 30–41.

[18] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. 9th ACM Symposium on Operating Systems Principles*, 2003, pp. 164–177.

[19] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: an overlay testbed for broad-coverage services," *ACM SIGCOMM Computer Communication Review*, vol. 33, pp. 3–12, July 2003.

[20] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*, 2003.

[21] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The NAS parallel benchmarks," *The International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[22] L. Deri and R. Andrews, "N2N: A layer two peer-to-peer VPN," in *Proc. 2nd Int. Conf. Autonomous Infrastructure, Management and Security: Resilient Networks and Services*, 2008, pp. 53–64.

[23] J. R. Lange and P. A. Dinda, "Transparent network services via a virtual traffic layer for virtual machines," in *Proc. 16th Int. Symposium on High Performance Distributed Computing*, 2007, pp. 23–32.

[24] CloudNet: http://www.cloud-net.org/.

[25] OpenVPN: http://openvpn.net/.