

WOW: Self-Organizing Wide Area Overlay Networks of Virtual Workstations

Arijit Ganguly, Abhishek Agrawal, P. Oscar Boykin, Renato Figueiredo
Advanced Computing and Information Systems Laboratory
University of Florida, Gainesville, Florida - 32611
email: {aganguly,aagraval,boykin,renato}@acis.ufl.edu

Abstract—This paper describes WOW, a distributed system that combines virtual machine, overlay networking and peer-to-peer techniques to create scalable wide-area networks of virtual workstations for high-throughput computing. The system is architected to: facilitate the addition of nodes to a pool of resources through the use of system virtual machines (VMs) and self-organizing virtual network links; to maintain IP connectivity even if VMs migrate across network domains; and to present to end-users and applications an environment that is functionally identical to a local-area network or cluster of workstations. We describe a novel, extensible user-level decentralized technique to discover, establish and maintain overlay links to tunnel IP packets over different transports (including UDP and TCP) and across firewalls. We also report on several experiments conducted on a testbed WOW deployment with 118 P2P router nodes over PlanetLab and 33 VMware-based VM nodes distributed across six firewalled domains. Experiments show that the latency in joining a WOW network is of the order of seconds: in a set of 300 trials, 90% of the nodes self-configured P2P routes within 10 seconds, and more than 99% established direct connections to other nodes within 200 seconds. Experiments also show that the testbed delivers good performance for two unmodified, representative benchmarks drawn from the life-sciences domain. The testbed WOW achieves an overall throughput of 53 jobs/minute for PBS-scheduled executions of the MEME application (with average single-job sequential running time of 24.1s) and a parallel speedup of 13.5 for the PVM-based fastDNAMl application. Experiments also demonstrate that the system is capable of seamlessly maintaining connectivity at the virtual IP layer for typical client/server applications (NFS, SSH, PBS) when VMs migrate across a WAN.

I. INTRODUCTION

Virtualization techniques address key challenges in the deployment of wide-area distributed computing environments. System virtual machines [46] such as VMware [51] and Xen [12] fully decouple the execution environment exposed to applications within a “guest” VM from that of its “host”, allowing nodes distributed across multiple domains to be configured and managed with a consistent software base [24], [35], [44]. Virtual networks ([29], [52], [33], [57]) decouple the management of address spaces and can provide full TCP/IP connectivity for nodes behind network address translation (NAT) and firewall routers. This paper describes a distributed system that integrates virtual machine and self-organized virtual networking to create scalable wide-area networks of virtual workstations (WOWs).

The system described in this paper is architected to: (1) scale to large number of nodes, (2) facilitate the addition

of nodes to the system through self-configuration of virtual network links, (3) maintain IP connectivity even if virtual machines migrate across domains, and (4) present to end-users and applications an environment that is functionally identical to a local-area network or cluster of workstations [8]. By doing so, WOW nodes can be deployed independently on different domains, and WOW distributed systems can be managed and programmed just like local-area networks, reusing unmodified subsystems such as batch schedulers, distributed file systems, and parallel application environments that are very familiar to system administrators and users.

While WOW techniques can be used in a variety of scenarios, the design has been driven by needs that have arisen in collaborative environments. Specifically, the techniques presented in this paper allow users of a community to easily pool together resources to form a WOW configured as a typical cluster — i.e. where nodes are configured with a homogeneous software distribution and are connected by a private network. Adding a VM “guest” hosted by a typical Linux/Windows x86-based platform to an already-running WOW involves a simple one-time setup by a local system administrator that requires minimal changes to the host: install a suitable VM monitor, clone a base VM image, and instantiate it.

Because WOWs run unmodified operating systems and applications inside VMs, they can be applied to facilitate the deployment of existing, successful middleware frameworks and reach a wide variety of hosts. For instance, a base WOW VM image can be installed with Condor [39] binaries and be quickly replicated across multiple sites to host a homogeneously configured distributed Condor pool, and a VM image installed with Globus [28] binaries and configured with a network interface to the public network can serve as a gatekeeper to this pool.

There are two key contributions in this paper. First, we describe a novel, extensible user-level framework to discover, establish and maintain overlay links to tunnel IP packets over different transports (including UDP and TCP) in a decentralized manner. Second, we report on several experiments conducted on a realistic deployment which demonstrate that the current implementation of WOW supports O/S and middleware configurations typical of cluster environments and provides good performance for high-throughput sequential jobs and for parallel applications.

On the first contribution, we have extended the Brunet

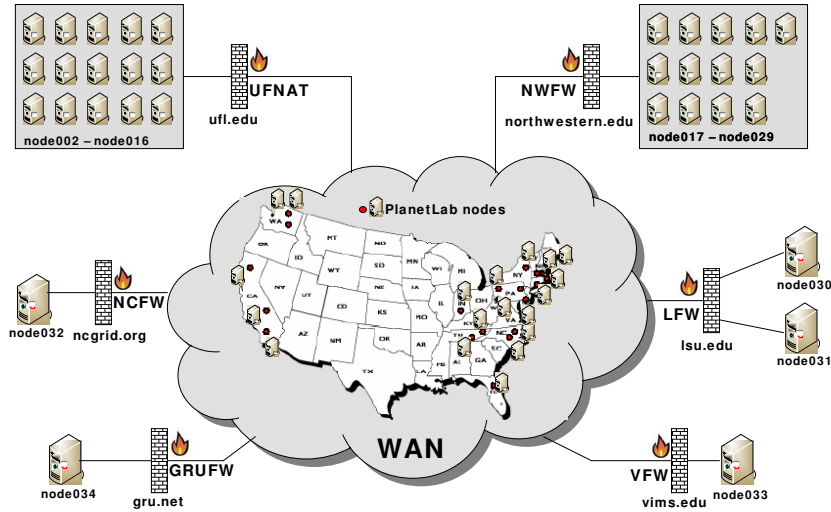


Fig. 1. WOW testbed used for experiments. The WOW has 33 compute nodes, 32 of which are hosted in universities and behind at least one level of NAT and/or firewall routers: 16 nodes in Florida; 13 in Illinois (Northwestern U.); 2 in Louisiana; and 1 node each in Virginia and North Carolina (VIMS and UNC). Node 34 is in a home network, behind multiple NATs (VMware, wireless router, and ISP provider). A total of 118 P2P router nodes which run on 20 PlanetLab hosts are also part of the overlay network.

P2P protocol [14] and the IPOP virtual network system [29] to support on-demand establishment of direct overlay links between communicating WOW nodes. Such direct connections allow nodes to bypass intermediate overlay routers and communicate directly with other nodes if IP packet exchanges between nodes are detected by the overlay. This technique is similar to the topology adaptation as described in [53], but we achieve the same in a purely decentralized manner that is capable of traversing NAT/firewall routers using hole-punching techniques [43], [25], [31] also in a decentralized manner.

On the second contribution, we have evaluated the performance of a WOW prototype quantitatively using a variety of experiments. The prototype uses the IPOP IP-over-P2P overlay [29] with support for decentralized establishment of direct connections, and VMware-based hosted VMs. The experiments are conducted in a testbed with 118 router nodes deployed on PlanetLab [20] and 33 compute nodes deployed across six different firewalled domains (Figure 1). The experiments are designed to (1) demonstrate the ability of our implementation to support existing middleware and compute-intensive applications, and (2) to quantify the performance benefits from direct overlay connections. Related publications have investigated the overhead of machine and network virtualization from several perspectives [51], [12], [24], [52], [33], [35], [44], [57].

Experimental results show that nodes joining the WOW can autonomously establish P2P connections and IP routability typically within 10 seconds, and establish direct connections with nodes they communicate within 200 seconds. Performance analyses using realistic batch-sequential and parallel applications show that the 33-node compute cluster sustains good throughput for batch jobs (53 jobs/minute for MEME [11] sequential runs with average run time of 24 sec-

onds) and delivers parallel speedups of 13.5x for fastDNAmI-PVM [41], [48]. The support for decentralized forming of direct connections provides a 240% higher job throughput for MEME and 24% better parallel performance for fastDNAmI-PVM, compared to a setup where direct connections are not supported.

We also report on the ability of WOW to autonomously re-establish virtual network links after a VM migrates across a wide-area network. These experiments show that WOW nodes successfully resume the execution of TCP/IP client/server applications after the migration (including a PBS-scheduled job which uses data from an NFS-mounted file system, and an SSH-based file transfer) in a manner that is completely transparent to the applications.

The remainder of this paper is organized as follows. Section II discusses related work. Section III describes the configuration and deployment of WOW virtual machine nodes. Section IV details the connection establishment techniques used to form virtual network links among WOW nodes. Section V describes the experimental setup and presents analyses of the performance of a WOW prototype deployment. Section VI presents directions for future work and concludes the paper.

II. RELATED WORK

Over the years, several related approaches have investigated high performance distributed computing using commodity computers and networks. The Berkeley NOW project [8] and Beowulf [13] are early and very successful efforts in this direction; today, the use of commodity clusters for high performance computing is widely disseminated. WOW shares the motivation of building systems of commodity machines and networks; in our case, the commodity machines are virtualized, and the commodity network is a self-configuring IP-over-P2P overlay [29] on top of the Internet. Rather than

supporting tightly-coupled parallel computation within a local-area or cluster network, the aim of WOW is to support high-throughput computing and cross-domain collaborations.

The Albatross [36] project aims to study applications and programming environments for wide-area cluster computers. In Cluster-on-Demand [18] Moore et.al propose to partition a cluster into dynamic virtual clusters with independent installed software, name spaces, access controls, and network storage volumes. The DAS [23] project built a distributed cluster based on homogeneously configured commodity nodes across five Dutch universities. Also related to our work is the Ibis project [22], [10] which lets applications span multiple sites of a grid, and copes with firewalls, local IP addresses, secure communication, and TCP bandwidth problems.

Several efforts on large-scale distributed computing have focused on aggregating wide-area resources to support high-throughput computing, but at the expense of requiring applications to be designed from scratch [7], [40], [15], [2], [1]. Legion [30] is a system also designed to scale to large numbers of machines and cross administrative domains. Globus [28] provides a security infrastructure for federated systems and supports several services for resource, data and information management. Condor [39], [42], [55] has been highly successful at delivering high-throughput computing to large numbers of users.

Our work differs from these approaches in that it is an end-to-end approach to providing a fully connected wide-area cluster environment for running unmodified applications. In WOW, the use of virtual machines and networks enables grids that run arbitrary applications without being tied to any particular application or middleware framework, and in that sense it is fundamentally different from the above systems. Nonetheless, the use of virtualization makes our approach one that does not preclude the use of any of these systems. Quite the contrary, because virtualization enables us to run unmodified systems software, we can readily reuse existing, mature middleware implementations when applicable, and rapidly integrate future techniques.

Researchers have also started applying P2P techniques to computational grids and wide area clusters [27]. In [19] Cheema et.al and in [32] Iamnitchi et.al have investigated P2P discovery of computational resources for grid applications. In [16] Cao et.al. have proposed a P2P approach to task scheduling in computational grids. Related to our work are the Jalapeno [56], Organic Grid [17], OurGrid [9] and ParCop [6] projects which also pursue decentralized computing using P2P technology. Our system currently applies P2P techniques to solve a different problem, which is the self-configuration of virtual network links to enable efficient and easy-to-deploy virtualized clusters.

The use of P2P-based overlays to support legacy applications has also been described in the context of i3 [49], [34]. The i3 approach is to use the P2P DHT system Chord[50] to redirect packets, potentially to mobile hosts and NATed hosts. The main difference in our approach is that we are providing a virtual network layer which must exactly emulate IP in order to

run existing IP-based grid software. By contrast, i3 introduces new programming APIs for nodes to communicate. The i3 project has developed proxies for UDP and TCP traffic. In contrast, our approach is to assign virtual IP addresses so that *all* IP traffic is communicated and the node with the virtual IP address works *exactly* the same as would on any IP network. In addition, the virtual network provided by WOW is completely *isolated* from the physical network.

Related projects (VIOLIN [33], VNET [52][53], ViNe [57]) have also recognized the utility of network overlays in wide area distributed environments. Our approach is fundamentally different in that nodes can join and leave the overlay in a completely decentralized, self-organizing fashion. In contrast, in VNET and VIOLIN it is necessary for administrators to set up overlay links, and a centralized entity is needed to orchestrate network adaptation in [53].

III. WOW NODE CONFIGURATION AND DEPLOYMENT

A. Background and Motivations

Commodity machines connected by local-area networks are very flexible and cost-efficient resources to run high-throughput computing and parallel applications. Scaling beyond a LAN, Grid efforts address important issues that arise when resources are shared among “virtual organizations” [28]. At the resource level, system configuration heterogeneity and the difficulty to establish connectivity among machines due to the increasing use of NATs/firewalls [47] substantially hinders sharing of resources. WOWs are designed to facilitate the aggregation of resources in an environment where systems in different domains have different hardware and software configurations and are subject to different machine and network administration policies. Resources aggregated as WOWs can then be managed by existing, unmodified Grid middleware (e.g. for scheduling and access control), allowing each site to specify usage policies as necessary.

Virtualization allows for isolated, flexible and efficient multiplexing of resources of a shared, distributed infrastructure [24]. With the use of VMs, the native or preferred software environment for applications can be instantiated on any physical host. VMs can be replicated to form virtual clusters [58], enabling the instantiation of homogeneous execution environments on top of heterogeneous systems. VMs offer unique opportunities for load balancing and fault tolerance that build upon growing support for checkpointing and “live” migration of running VMs in monitors such as in Xen [21]. Complementary to VMs, virtual networking enables isolated multiplexing of private networks with a familiar TCP/IP environment for communication among nodes [52], [33].

B. VM configuration

WOWs are deployed in a manner that shares common characteristics with cluster computing environments, thus configuring and deploying a WOW is a process that system administrators are for the most part already familiar with. A base computing node is configured with a desired execution environment, e.g. a particular Linux distribution and ancillary

libraries, system utilities and application software. Each node is an independent computer which has its own IP address on a private network. While in a typical LAN cluster each physical node’s disk is loaded with the desired execution environment and nodes within the cluster rack are connected by hardware switches, in WOW a virtual disk is configured and copied to all hosts, and nodes across a WAN are interconnected by a software overlay.

Our system works with VMs that provide a NAT-based virtual network interface, such as VMware and Xen, and does not require the allocation of a physical IP address to the VM. The only software needed within the VM that is additional to a typical cluster environment is the IPOP [29] virtual network. The current implementation of IPOP requires only the following components: the mono .NET runtime environment, a “tap” device, and a short (tens of lines) configuration script to launch IPOP and setup the VM with an IP address on the overlay. The configuration script specifies the location of at least one IPOP node on the public Internet to establish P2P connections with other nodes. Currently, we use an overlay deployed on PlanetLab for this purpose.

C. Deployment and usage scenarios

Our goal is to make the addition of a node to a pool of Grid resources as simple as instantiating a pre-configured VM image that can be easily disseminated by file transfer or through distribution of physical media. We envision WOW deployments where a VM “appliance” is configured once, then copied and deployed across many resources, facilitating the deployment of open environments for grid computing similar in nature to efforts such as the Open Science Grid (OSG [3]). WOW allows participants to add resources in a fully decentralized manner that imposes very little administrative overhead.

High-throughput computing workloads are a primary target of our system because they can benefit from the large number of nodes which WOW provides, regardless of differences in their speeds. Loosely-coupled parallel applications can also benefit from WOW. Our architecture also is designed to facilitate the setup of collaborative environments, for example those that support large numbers of users who execute both interactive and batch applications [26], [5].

IV. WOW OVERLAY CONNECTION MANAGEMENT

At the core of the WOW architecture is the ability to self-configure overlay links for nodes that join the distributed system. This section describes the adaptive virtual IP networking used for WOW. This work builds upon a user-level framework provided by the Brunet P2P protocol suite [14]. We have extended Brunet and the IPOP virtual networking system [29] to discover, establish and maintain direct (single-hop) overlay links between frequently communicating nodes. We use the term “connection” to refer to an overlay link between P2P nodes over which packets are routed. In this section, we describe the different connection types supported in Brunet and protocols for connection setup. In Section IV-E, we describe our adaptive algorithm to support direct P2P

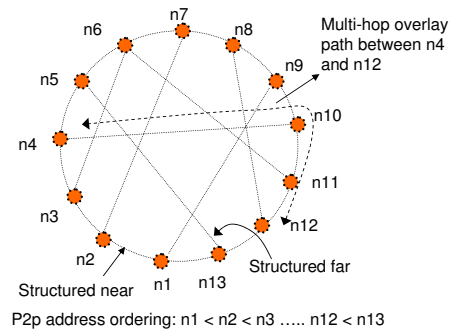


Fig. 2. Peer-to-peer connections in Brunet. Nodes always form structured near connections with neighboring nodes of a ring, and structured far connections are created by Brunet to reduce the average number of overlay hops.

connection establishment between communicating nodes so that they can communicate over a single overlay hop. We refer to such direct connections as *shortcut* connections. Shortcut connections alleviate issues of geography and load unaware P2P routing, described in [29].

In this section we describe the P2P system and how it handles overlay network connections. In particular, we describe how our connection and linking protocols enable new nodes to join the network, connections to be formed between nodes behind NATs, and decentralized creation of *shortcut* connections based on traffic inspection.

A. Description of the P2P System

Brunet maintains a structured ring of P2P nodes ordered by 160-bit Brunet addresses (Figure 2). Each node maintains connections to its nearest neighbors in the P2P address space called *structured near* connections. When a new node joins the P2P network, it must find its right position on the existing P2P ring and form *structured near* connections with its nearest neighbors in the P2P address space. From that point onwards, the node can communicate with every other node in the P2P network and vice versa. Each node also maintains k connections to distant nodes in the P2P address space called *structured far* connections, which reduces the average number of overlay hops between nodes to $O(\frac{1}{k} \log^2(n))$ for a network of size n using the algorithm of [37].

Brunet uses greedy routing of packets over structured (near and far) connections, where at each overlay hop the packet gets closer to the destination in the P2P address space. The packet is eventually delivered to the destination; or if the destination is down, it is delivered to its nearest neighbors in the P2P address space.

Connections between Brunet nodes are abstracted and may operate over any transport. The information about transport protocol and the physical endpoint (e.g. IP address and port number) is contained inside a Uniform Resource Indicator (URI), such as *brunet.tcp:192.0.1.1:1024*. Note that a P2P node may have multiple URIs, if it has multiple network interfaces or if it is behind one or more levels of network

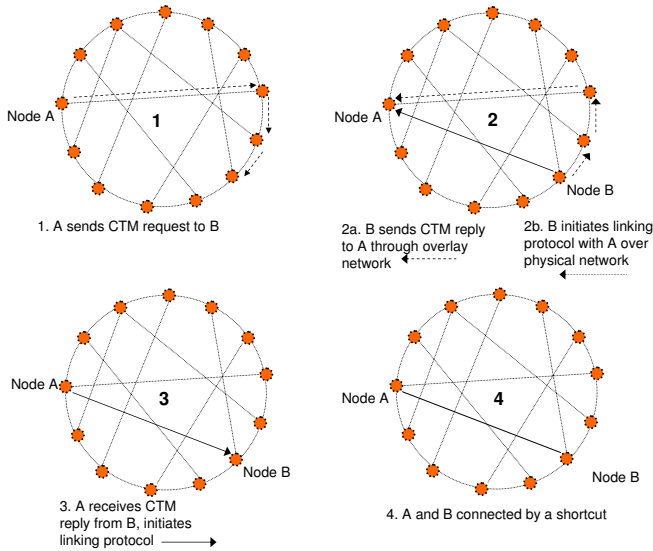


Fig. 3. Shortcut connection setup between nodes A and B. 1: A initiates the process with Connect-to-me (CTM) request to B. 2: B send a CTM reply through the overlay, and initiates the linking protocol directly with A, the URI list received in step 1. 3: upon receiving a CTM reply, A starts the linking protocol handshake using B's URI list. 4: a one-hop connection between A and B is established by the linking protocol.

address translation. The encapsulation provided by URIs provides extensibility to new connection types; currently there are implementations for TCP and UDP transports.

B. Connection setup

Figure 3 shows the steps involved in connection setup between two P2P nodes. The mechanism for connection setup between nodes consists of conveying the intent to connect, and resolution of P2P addresses to URIs followed by the linking handshake, which are summarized as follows:

- 1) *Connection protocol*: Through the connection protocol, nodes convey their intent to setup a connection and their list of URIs to another P2P node. A node that wishes to connect to another node in the network sends a *Connect To Me* (CTM) request message to that node. The CTM request contains the list of URIs of the initiating node. The message is routed over the P2P network and delivered to the target node, which in turn sends a CTM reply (containing its own URIs) back to the source node, and also initiates the linking protocol which we describe next. The source node on getting the CTM reply initiates the linking protocol with the target node. It should be noted that the target node for a prospective connection may not be up, in which case the CTM request is delivered to its nearest neighbors which then become the target for linking.
- 2) *Linking protocol*: A P2P node initiates the linking protocol when it gets a CTM request or when it gets a reply for the CTM request it sent out. In either case, it has already learned the URIs for the node on the other side of the upcoming connection. The node starts the

linking handshake by sending a sequence of link request messages to the target node over the physical network trying the URIs it has learnt one by one. The target node simply responds with link reply messages over the physical network. Following exchange of sequence of linking handshake messages, the two nodes record the new connection state which can subsequently be used for routing packets over the network.

Nodes keep an idle connection state alive by periodically exchanging *ping* messages, which also involves resending of unresponded pings and exponential back-offs between resends. Any unresponded ping message is perceived as the node going down or network disconnection, and the node discards the connection state. These ping messages incur bandwidth and processing overhead at end nodes, which restricts the number of connections a node can maintain.

It should be noted that the linking protocol is initiated by both the peers, leading to a potential race condition that must be broken in favor of one peer succeeding while the other failing. Therefore, each node records its *active* linking attempt to the target, before sending out a link request. Now, if the current node now gets a link request from its target, it responds with a link error message stating that the target should give up its *active* attempt and let the current node go ahead with the protocol. The target node gives up its connection attempt, and eventually the current node would succeed. It is possible that both the nodes (current and target) initiate *active* linking, get link error messages from the other and give up their attempts, in which case they restart with an exponential backoff, to reduce the likelihood of another race condition.

C. Joining an existing network and acquiring connections

A new P2P node is initialized with URIs of a few nodes already in the network. The new node creates what we call a *leaf* connection with one of these initial nodes by directly using the linking protocol. In this case the linking is unidirectional and the target passively participates in the protocol. These initial nodes are typically public and if the new node is behind a NAT, it discovers and records its own URIs corresponding to the NAT assigned IP/port. Once the leaf connection is established, the *leaf* target (which is already in the ring) acts as forwarding agent for the new node. At this stage, there is no guarantee that any packet originating at an arbitrary point in the network and addressed to the new node will be delivered to it.

The new node must now identify its correct position in the ring, and form *structured near* connections with its left and right neighbors. It sends a CTM request addressed to itself on the network through the *leaf* target. The CTM request is routed over the structured network, and (since the new node is still not in the ring) eventually delivered to its nearest neighbors. The CTM replies received by the forwarding node are passed back to the new node. The node now knows the URIs of its nearest (left and right) neighbors and vice versa, and can form *structured near* connections with them. At this point, the node is fully routable. We have measured the time taken from a new

node joining an existing P2P network of more than 130 nodes, to it becoming fully-routable and found it is of the order of seconds (see Figure 4).

Once the node is at its right place in the ring, it acquires *structured far* connections to random¹ nodes on the ring using protocols described in Section IV-B.

D. NAT traversal

Based on the description of URIs presented earlier, it follows that a node inside a private network behind a NAT can have multiple URIs (corresponding to the private IP/port, and NAT assigned IP/port when it communicates with nodes on public internet). Initially, nodes behind NATs only know their private IP/port, and during connection setup with public nodes they also learn their NAT-assigned IP/port. Furthermore, not all URIs can be used to communicate with it. Which URIs are usable depend on the locations of communicating nodes and the nature of NATs. For example, two nodes behind a NAT that does not support “hairpin” translation [25] can communicate only using URIs corresponding to their private IP/port, and they fail when using the NAT assigned IP/port. In contrast, two nodes behind “hairpin” NATs can communicate using both URIs (private and NAT assigned). A public node can communicate with a node inside a private network only using the URI corresponding to the NAT assigned IP/port.

The UDP transport implementation of Brunet is designed to deal with NAT traversal for large class of NAT devices found in practical deployments. The bi-directionality of the connection/linking protocols is what enables the NAT hole-punching [43], [25], [31] technique to succeed. As described earlier, a P2P node (in a private network behind a NAT) may have multiple URIs, and due to firewalls and NATs some nodes might be unaddressable on some of their URIs. During the linking protocol, nodes try each others URIs one at a time, until they find one over which they can send and receive handshake messages. The linking handshake involves the resending (with exponential back-off of the resend interval) of link requests that are not responded within a certain interval. If a node does not get responses for link requests it sends out even after a maximum number of retries, it restarts the linking handshake over the next URI in the list, until it eventually succeeds or gives up².

E. Adaptive shortcut creation

In [29], latencies as high as 1600 ms were observed between IPOP nodes connected to a 100 node PlanetLab network. The high latencies were due to multi-hop overlay routing through highly loaded PlanetLab nodes. This section describes our method for decentralized adaptive shortcut creation which enables the setup of single-hop overlay links on demand, based on traffic inspection. As we will show in Section V,

¹The logic for choosing the random address is out of scope of this paper.

²Currently, the back-off factor and number of retries have been chosen conservatively in Brunet to account for highly loaded nodes in environments such as PlanetLab, which can lead to delays of the order of 150 seconds before giving up on a bad URI and trying the next one.

shortcuts greatly reduce latency and improve bandwidth of the virtual network, as well as improve job throughput of the grid compute nodes.

The Brunet P2P library is an extensible system which allows developers to add new routing protocols and connection types. In Brunet, for each connection type, each P2P node has a *connection overlord* which ensures the node has the right number of connections.

To support shortcut P2P connections, we have implemented a *ShortcutConnectionOverlord* within the Brunet library. The *ShortcutConnectionOverlord* at a node tracks communication with other nodes using a metric called *score*. Our algorithm is one based on a queueing system. The number of packets that arrive in the i^{th} unit of time is a_i . There is a constant service rate on this work queue. The score is the amount of remaining work left in this virtual queue. If the score at time i is s_i , and the rate of service is c , we have:

$$s_{i+1} = \max(s_i + a_i - c, 0)$$

The higher the *score* of a destination node, the more communication there has been with it. The nodes for which the virtual queue is the longest are the nodes we connect to. The *ShortcutConnectionOverlord* establishes and maintains shortcut connections with nodes whose *scores* exceed a certain threshold.

Although it is conceivable to create shortcuts to every other node in the network, for large overlays the bandwidth and processing overhead incurred during connection setup and maintenance poses a practical limit on the number of shortcuts a node can maintain. In future work we plan to investigate models capturing the relationship between number of shortcut links and the cost associated with maintaining them (processing and bandwidth) to better select the score threshold, which is currently a constant.

In the next section we present our experimental results which compare bandwidth and latency of the virtual network with and without our adaptive shortcuts. We also measure the time required for the network to adapt and create shortcuts between communicating nodes.

V. EXPERIMENTS

A. Experimental setup

We deployed a virtual cluster of 33 VMs configured with the same Debian/Linux based O/S, thus providing a homogeneous software environment within the cluster. These VMs were instantiated on top of a highly heterogeneous physical environment, consisting of hosts running different operating systems (Linux and Windows) and different VM monitors, located in different network domains, and subject to different firewall policies. Table I details the configuration of the various compute nodes of the testbed illustrated in Figure 1. The 118 P2P nodes on the PlanetLab network were used to provide a “bootstrap” overlay running on public-address Internet nodes, to which nodes behind firewalls could connect. The PlanetLab nodes run the IPOP software for routing, but do not have “tap” virtual network interfaces attached to pick/inject packets

TABLE I
CONFIGURATION OF THE WOW TESTBED DEPICTED IN FIGURE 1. ALL WOW GUESTS RUN THE SAME DEBIAN/LINUX 2.4.27-2 O/S.

ode number	Virtual IP (172.16.1.x)	Physical Domain	Host CPU	Host O/S	VM monitor (VMware)
node002	.2	ufl.edu	Xeon 2.4GHz	Linux 2.4.20-20.7smp	Workstation 5.5
node003-node016	.3 - .16	ufl.edu	Xeon 2.4GHz	Linux 2.4.20-20.7smp	GSX 2.5.1
node017-node029	.17 - .29	northwestern.edu	Xeon 2.0GHz	Linux 2.4.20-8smp	GSX 2.5.1
node030-node031	.30 - .31	lsu.edu	Xeon 3.2GHz	Linux 2.4.26	GSX 3.0.0
node032	.32	ncgrid.org	Pentium III 1.3GHz	Linux 2.4.21-20.ELsmp	VMPlayer 1.0.0
node033	.33	vims.edu	Xeon 3.2GHz	Linux 2.4.31	GSX 3.2.0
node034	.34	gru.net	Pentium 4 1.7GHz	Windows XP SP2	VMPlayer 1.0.0

from/into the host³. The IPOPOP prototype is based on the Brunet P2P library; it is written in C#, and the run-time environment is based on mono 1.1.9.2. Brunet can be configured to establish UDP or TCP links to tunnel the virtual network traffic; in this paper, we have used UDP.

With the only exception of the ncgrid.org firewall, which had a single UDP port open to allow IPOPOP traffic, no firewall changes needed to be implemented by system administrators. Furthermore, WOW nodes used VMware NAT devices, which do not require an IP address to be allocated by the site administrator.

The remaining of this section analyzes the performance of our testbed from several perspectives. Throughout this section, for brevity's sake, we refer to the Northwestern University site as "NWU" and to the University of Florida site as "UFL".

B. Shortcut connections: Latency and Bandwidth

An initial analysis of the latency and throughput of IPOPOP links in LAN and WAN environments has been presented in previous work [29]. In this experiment we study the process of joining an IPOPOP node to an existing overlay network using the following experiment. An IPOPOP node "A" was instantiated a priori at UFL with a virtual IP address that remained fixed during the experiment (172.16.1.2). We then proceeded an iterative process of: (1) starting up an IPOPOP node "B" on a host at NWU; (2) sending 400 ICMP echo packets from B to A at 1 second intervals; and (3) terminating the IPOPOP node B. This process was repeated for 10 different virtual IP addresses (mapping B to different locations on the P2P ring), with 10 experiments conducted for each IP address, resulting in a total of 100 trials. Experiments were also conducted for two other scenarios: both nodes A and B at NWU, and both nodes at UFL.

Figure 4 summarizes the results from this experiment. Focusing initially on the UFL-NWU case, analyzing the data for the initial tens of ICMP packets shows three different regimes (see Figure 5). For the first three ICMP requests, on average, 90% of the packets are dropped⁴. Within this short period, node B is unlikely to establish a route to other nodes. Between ICMP sequence numbers 4 and 32, the average

percentage of lost packets steadily drops to below 1%, and the average round-trip latency is also drops from 146ms to 43 ms, with a standard deviation of 4.9 ms. These values indicate that during this period the newly joined node is very likely to be routable, and may have also established a shortcut connection with node A.⁵

Finally, between ICMP sequence numbers 33 and 400, the percentage of lost packets drops to 1%, and the round trip latencies drop to 38ms (standard deviation of 1.3ms), indicating that a shortcut connection has been established.

In the UFL-UFL case, we observe the same three regimes; however, the timings differ. It takes up to about 40 ICMP ping packets before the node becomes routable over the P2P network. Furthermore, it takes about 200 ICMP packets before shortcut connections are formed. This high delay is because of the nature of the UFL NAT and the current implementation of the IPOPOP linking protocol, as follows. The UFL NAT does not support "hairpin" translation [25], i.e. it discards packets sourced within the private network and destined to the NAT-translated public IP/port. As described in Section IV-B, the linking handshake involves nodes trying target URIs one by one until finding one on which they can send and receive handshake messages. In IPOPOP, nodes first attempt the URIs corresponding to the NAT assigned public IP/port for the linking handshake during the connection setup. Because of conservative estimates for the re-send interval, the back-off factor and the number of retries for UDP tunneling, nodes take several seconds before giving up on that URI and trying the next in the list (private IP/port) on which they succeed.

For the NWU-NWU case, the two nodes are either inside the same private network (the VMware NAT network), or on different hosts. The VMware NAT supports hairpin translation, hence both URIs for the P2P node work for connection setup. As with the UFL-NWU case, the linking protocol succeeds with the first URI it tries and hence the short cut connections are setup within a few (about 20) ICMP packets.

We also evaluated the bandwidth achieved between two WOW nodes communicating over the virtual network provided by IPOPOP. We used the Test TCP (ttcp) utility to measure the end-to-end bandwidth achieved in transfers of large files. The

³We used PlanetLab router nodes to investigate the effectiveness of our approach for a large overlay network. WOWs can also be deployed such that overlay routing is distributed across IPOPOP running on compute nodes.

⁴Due to the y-axis scale chosen for Figure 4, these few initial packets do not appear on the plot.

⁵For each trial of this experiment, we observed a sharp fall in ICMP latency as soon as a shortcut connection was established. However, the time elapsed before the shortcut formed varied over the length of this regime for different trials; averaging over 100 such trials shows the gradual decline in latency and likelihood of packets being dropped.

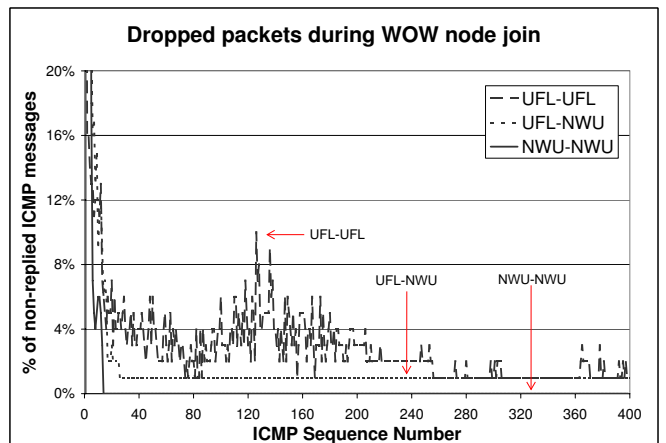
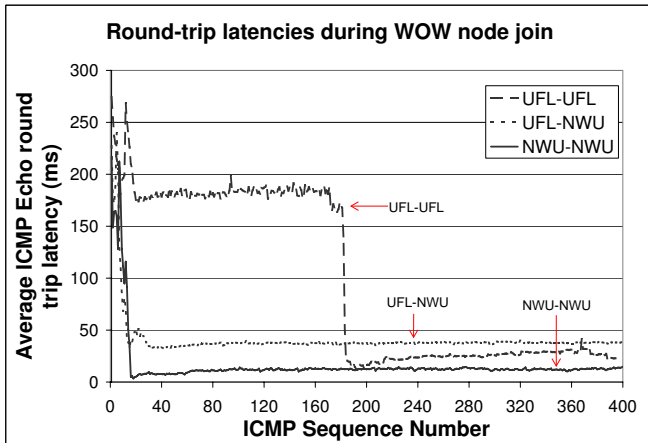


Fig. 4. Profiles of ICMP echo round-trip latencies (left) and dropped packets (right). The experiment considers three combinations of the location of the node A joining the network and the node B it communicates to: UFL-UFL, UFL-NWU and NWU-NWU. Left: The plot shows latencies averaged over 100 trials as reported by the ping application for packets which were not dropped. Right: The plot shows the percentage of lost packets (over 100 trials) for each ICMP sequence number as reported by the ping application.

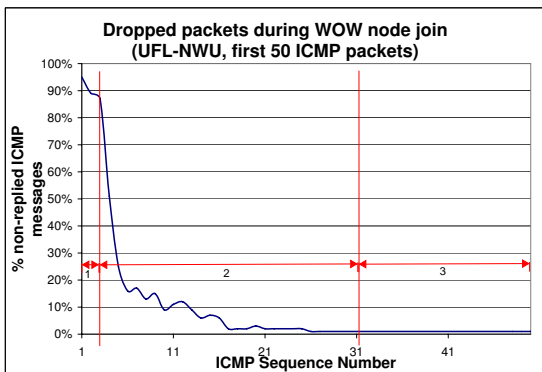


Fig. 5. The three regimes for percentage of dropped ICMP packets observed at new node B. (1) The new node is not routable over the P2P network. (2) The node is routable and may have formed a short-cut connection to node A. (3) The new node has formed a short-cut connection with node A.

results are summarized in Table II. Two factors along the routing path limit the bandwidth between two nodes: first, the bandwidth of the overlay links, and second, the load of machines hosting the intermediate IPOP routers, which reduces the processing throughput of our user-level implementation. Without shortcut connections, nodes communicated over a 3-hop communication path traversing the heavily loaded PlanetLab nodes and a very low bandwidth was recorded. However, with shortcuts enabled, nodes communicate over a single overlay hop, thus achieving a much higher bandwidth.

C. Virtual machine migration

VMs provide a unique opportunity to migrate unmodified applications between hosts [38], [45], [21]. However, when a VM migrates it also carries along its connection state. Such connection state can also accumulate inside other hosts with which it is communicating. This forces the VM to retain its network identity, which in turn hampers VM migration between subnets. Virtual networking provides the opportunity

TABLE II
AVERAGE BANDWIDTH AND STANDARD DEVIATION MEASUREMENTS BETWEEN TWO WOW NODES: WITH AND WITHOUT SHORTCUTS. THE EXPERIMENT CONSIDERS 12 TTCP-BASED TRANSFERS OF FILES OF THREE DIFFERENT SIZES (695MB, 50MB AND 8MB) AND TWO SCENARIOS FOR THE LOCATION OF NODES: UFL-UFL AND NWU-UFL.

	Shortcuts enabled		Shortcuts disabled	
	Bandwidth KB/s	Std.dev KB/s	Bandwidth KB/s	Std.dev KB/s
UFL-UFL	1614	93	84	3
UFL-NWU	1250	203	85	2.3

of maintaining a consistent network identity for a VM, even when it migrates to a different network.

The network virtualization layer of WOW is designed such that, when a VM migrates to another subnet, the connection state for the *virtual* network interface continues to remain valid. However, the *physical* network state involving the P2P overlay connections needs to be invalidated. In the case of IPOP, this can be done in a very simple manner — by simply killing and restarting the user-level IPOP program. The IPOP node then rejoins the overlay network autonomously, creating structured connections on the P2P ring following the same process described in Section IV. Clearly, packets do not get routed and are dropped until the node rejoins the P2P network; this short period of no routability is approximately 8 minutes for the 150-node network used in our setup. The TCP transport and applications are resilient to such temporary network outages, as the following experiments show.

1) *SSH/SCP file transfer*: In this experiment, we set up a client VM with a VMware NAT interface at NWU. The file server was located in the UFL private network. The client VM established an SSH/SCP-based file transfer session with the server, and started to download a 720MByte file. During the transfer, at around 200s of elapsed transfer time, the migration was initiated: the IPOP process on the file server was killed,

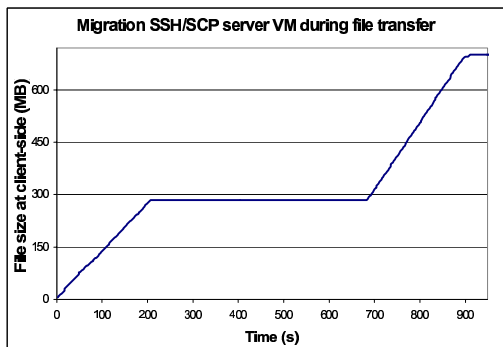


Fig. 6. Profile of the size of a 720MB file transferred to an SSH/SCP client during the migration of the SSH/SCP server. The y-axis shows the file size of the client’s local disk. The client VM is at NWU; the server VM is suspended at the UFL and resumed at NWU. The file transfer resumes successfully without requiring any application restarts. The sustained transfer bandwidths before and after migration are 1.36MB/s and 1.83MB/s, respectively.

the VM was suspended, its virtual memory and disk copy-on-write logs were transferred to the host at NWU, and the VM was then resumed.

When the VM was resumed, its virtual eth0 network interface was restarted, and because the NATs that the VM connected to were different at UFL and NWU, the VM acquired a new physical address for eth0 at the destination. However, the virtual tap0 interface did not need to be restarted and remained with the same identity on the overlay network. Then IPOP was restarted; seconds later, the SCP server VM again became routable over the P2P network, then established a shortcut connection with the SCP client VM, and eventually the SCP file transfer resumed from the point it had stalled. Figure 6 summarizes the results from this experiment.

2) *PBS job submission*: In this experiment, we setup an environment for the migration of a VM which executes a sequential job scheduled by PBS. The PBS head node VM was configured in the UFL private network, and 2 worker VMs were configured on two different hosts also in the UFL private network. Communication within these nodes was performed over the IPOP virtual network.

This experiment simulates a use case of applying migration to improve load balancing: we introduced a background load to a VM host, observed an increase in the execution time of applications executing on the VM guest, and migrated the VM guest from UFL to a different host at NWU. IPOP was restarted on the guest upon VM resume. We observed that the job which was running on the VM continued to work without problems, and eventually committed its output data to the NFS-mounted home directory of the account used in the experiment. While the runtime for the job that was “in transit” during the migration was increased substantially due to the WAN migration delay, once PBS started submitting jobs to the VM running on an unloaded host we observed a decrease in job runtimes with respect to the loaded host. This experiment also showed that the NFS and PBS client/server implementations were tolerant to the period with lack of connectivity. Figure 7 summarizes the results from this experiment.

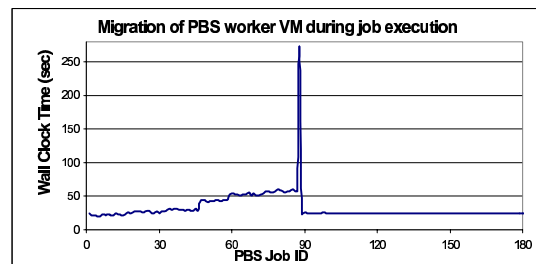


Fig. 7. Profile of execution times for PBS-scheduled MEME sequential jobs during the migration of a worker node. Job IDs 1 through 87 run on a VM at UFL. During job ID 88, the VM is migrated to NWU. Job ID 88 is impacted by the wide-area migration latency of hundreds of seconds, but completes successfully. Subsequent jobs scheduled by PBS also run successfully on the migrated VM, without requiring any application reconfiguration or restart.

D. Application experiments

For this experiment we chose two representative life-science applications as benchmarks: MEME [11] version 3.5.0 and fastDNAm1-p [41], [48] version 1.2.2. These applications ran, without any modifications, on the 33-node WOW; scheduling, data transfer and parallel programming run-time middleware also ran unmodified, including OpenPBS [4] version 2.3.16, PVM [54] version 3.4.5, SSH, RSH and NFS version 3.

The experiments were designed to benchmark our implementation for classes of target applications for WOW: high-throughput independent tasks and parallel applications with high computation-to-communication ratios. Specifically, the goals of the experiments are: (1) to show that WOWs can deliver good throughput and parallel speedups, (2) to quantify the performance improvements due to shortcut connections, and (3) to provide qualitative insights on the deployment, use and stability of our system in a realistic environment.

1) *PBS batch application*: MEME [11] is a compute-intensive application that implements an algorithm to discover one or more motifs in a collection of DNA or protein sequences. In this experiment, we consider the execution of a large number (4000) of short-running MEME sequential jobs (approximately 30s each) queued and scheduled by PBS. The jobs run with the same set of input files and arguments, and are submitted at a frequency of 1 job/second at the PBS head node. Jobs read and write input and output files to an NFS file system mounted from the head node.

For the scenario where shortcut connections were enabled, the overall wall-clock time to finish the 4000 jobs was 4565s, and the average throughput of the WOW was 53 jobs per minute. Figure 8 shows a detailed analysis of the distribution of job execution times, for both the cases where the WOW had shortcut connection establishment enabled (left) and disabled (right). The differences in execution times shown in the histogram can be qualitatively explained with the help of Table I: most physical machines in the WOW prototype have 2.4GHz Pentium-4 CPUs; a couple of them (nodes 32 and 34) are noticeably slower, and three of them are noticeably faster (nodes 30, 31 and 33). Overall, the slower nodes end up running a substantially smaller number of jobs than the

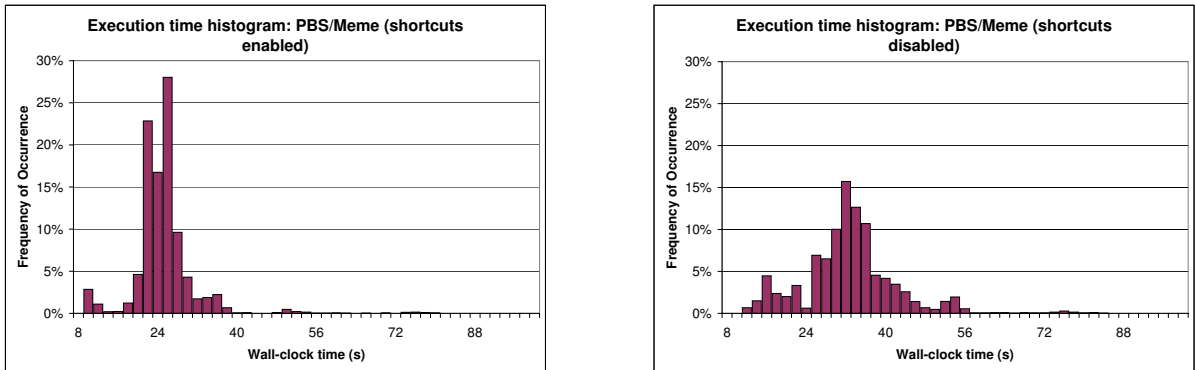


Fig. 8. Distributions of PBS/MEME job wall clock times when the overlay network self-organizing shortcuts are enabled (left) and disabled (right). The wall clock time average and standard deviation are 24.1s and 6.5s (shortcuts enabled) and 32.2s and 9.7s (shortcuts disabled).

TABLE III

EXECUTION TIMES AND SPEEDUPS FOR THE EXECUTION OF FASTDNAML-PVM IN 1, 15 AND 30 NODES OF THE WOW. PARALLEL SPEEDUPS ARE REPORTED WITH RESPECT TO THE EXECUTION TIME OF NODE 2.

	Sequential Execution		Parallel Execution		
	Node 2	Node 34	15 Nodes Shortcuts enabled	30 Nodes	
				Shortcuts disabled	Shortcuts enabled
Execution time (seconds)	22272	45191	2439	2033	1642
Parallel speedup (with respect to node 2)	n/a	n/a	9.1	11.0	13.6

fastest nodes (node 32 runs 1.6% of the jobs, while node 33 runs 4.2%).

Figure 8 also shows that the use of shortcut connections decreases both the average and the relative standard deviation of the job execution times. The use of shortcuts also reduced queuing delays in the PBS head node, which resulted in substantial throughput improvement, from 22 jobs/minute (without shortcut connections) to 53 jobs/minute (with shortcut connections).

Note that the throughput achieved by our system depends not only on the performance of the overlay, but also on the performance of the scheduling and data transfer software that runs on it (PBS, NFS). The choice of different middleware implementations running inside WOW (e.g. Condor, Globus) can lead to different throughput values, but comprehensive analysis and comparison of performance is beyond the scope of this paper. The VM technology in use also impacts performance. For MEME application, we observed an average virtual/physical overhead of 13%.

2) *PVM parallel application*: FastDNAML is a program for the maximum likelihood inference of phylogenetic trees from DNA sequence data [41], [48]. The parallel implementation of fastDNAML over PVM is based on a master-workers model, where the master maintains a task pool and dispatches tasks to workers dynamically. It has a high computation-to-communication ratio and, due to the dynamic nature of its task

dispatching, it tolerates performance heterogeneities among computing nodes.

Table III summarizes the results of this experiment. We used the same 50-taxa input dataset reported in [48]. The parallel execution of fastDNAML on the WOW reduces significantly the execution time. The fastest execution is achieved on 30 nodes when the WOW has shortcut connections enabled: 24% faster than 30 nodes without shortcuts enabled, and 49% faster than the 15-node execution. Although fastDNAML has a high computation-to-communication ratio for each task, the use of shortcuts resulted in substantial performance improvements. While we have not profiled where time is spent within the application during its execution time, increase in execution times can be explained by the fact that the application needs to synchronize many times during its execution, to select the best tree at each round of tree optimization [48].

The sequential execution times of fastDNAML are reported for two different nodes (node002 and node034) and show that the differences in the hardware configuration of the individual nodes of the WOW result in substantial performance differences. While modeling parallel speedups in such a heterogeneous environment is difficult, we report on the speedups with respect to a node which has the hardware setup most common in the network. The parallel speedup computed under this assumption is 13.6x; in comparison, the speedup reported in [48] is approximately 23x, but is achieved in a homogeneous

IBM RS/6000 SP cluster within a LAN.

E. Discussion

Our prototype WOW setup has been deployed and in use for more than a month. Here we discuss some of the qualitative insights that we have obtained from the practical usage of this system.

The deployment of the WOW was greatly facilitated by packaging all the software within the VM and requiring only a NAT network, as well as by the availability of free x86-based VM monitors, notably VMPlayer. Once a base VM image was created, replicating and instantiating new cluster nodes was quite simple. Using a NAT interface was convenient, but we believe that the use of a host-only interface will further improve the isolation of WOW nodes from the physical network, and we are working on an implementation that supports this interface.

The overlay network has been up and running since it was deployed for the experiments, and has been quite stable. In no occasion did we have to “restart” the entire overlay network, despite the fact that several physical nodes have been shut down and restarted during this period of time. The overlay network has also been resilient to changes in NAT IP/port translations, if and when they happen — a behavior we noticed in the VM hosted by a broadband-connected home desktop. IPOP dealt with these translation changes autonomously by detecting broken links and re-establishing them using the connection techniques discussed in Section IV.

VI. CONCLUSIONS AND FUTURE WORK

The functionality enabled by WOWs is very powerful, and our experimental results show that this is a viable technology for high throughput computing applications. There are several aspects of this system which call for better performance and more flexibility, and are the focus of on-going and future research. In terms of performance, the virtual network overhead is still relatively large in a local-area or cluster network. A significant component of this overhead is due to the multiple user/kernel copies that are involved in the handling of virtual network traffic. Solutions that support shared user/kernel buffers within WOW VMs (i.e. without requiring host kernel changes), and the use of high-speed remote DMA network interfaces to tunnel overlay traffic can reduce the amount of user/kernel copying to provide better LAN performance.

Our goal is to support substantially larger WOW setups than the one used in the performance evaluations. The overlay IP-over-P2P routing infrastructure of WOW is based on algorithms that are designed to scale to very large systems. The middleware that runs within the WOW for tasks such as scheduling and distributed file systems is often based on client/server models and may not scale to the same large numbers. In future work we plan to investigate approaches for decentralized resource discovery, scheduling and data management that are suitable for large-scale systems. We believe that WOWs provide a powerful and flexible infrastructure

that allows novel evolutionary and revolutionary middleware approaches to be deployed side-by-side with existing systems.

VII. ACKNOWLEDGMENTS

Effort sponsored by the NSF under grants EIA-0224442, EEC-0228390, ACI-0219925, ANI-0301108, SCI-0438246 and SCI-0537455 and carried out as a component of the “SURA Coastal Ocean Observing and Prediction (SCOOP) Program”, an initiative of the Southeastern Universities Research Association (SURA). Funding support for SCOOP has been provided by the Office of Naval Research, Award# N00014-04-1-0721 and by the NOAA Ocean Service, Award # NA04NOS4730254. The authors also acknowledge a SUR grant from IBM. 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 sponsors. The authors would like to thank Peter Dinda, Justin Davis, Vladimir Paramygin, Chirag Dekate, David Forrest and Steve Thorpe for providing access to host resources.

REFERENCES

- [1] distributed.net. <http://distributed.net/>.
- [2] fightaids@home. <http://fightaidsathome.scripps.edu/>.
- [3] Open science grid web site. <http://www.opensciencegrid.org>.
- [4] Openpbs web site. <http://www.openpbs.org>.
- [5] S. Adabala, V. Chadha, P. Chawla, R. J. Figueiredo, and J. A. B. Fortes et al. From virtualized resources to virtual computing grids: The INVIGO system. *Future Generation Computing Systems, special issue on Complex Problem-Solving Environments for Grid Computing*, 21(6), Apr 2005.
- [6] N. A. Al-Dmour and W. J. Teahan. Parcop: A decentralized peer-to-peer computing system. In *Proc. of the 3rd Intl. Symp. on Algorithms, Models and Tools for Parallel Computing on Heterogenous Networks*, Jul 2004.
- [7] D. P. Anderson, J. Cobb, E. Korpella, M. Lebofsky, and D. Werthimer. Seti@home: An experiment in public-resource computing. *Communications of the ACM*, 11(45):56–61, 2002.
- [8] T. E. Anderson, D. E. Culler, and D. A. Patterson et.al. A case for network of workstations: Now. *IEEE Micro*, February 1995.
- [9] N. Andrade, L. Costa, G. Germoglio, and W. Cirne. Peer-to-peer grid computing with the ourgrid community. In *Proc. of the 23rd Brazilian Symp. on Computer Networks*, May 2005.
- [10] O. Aumage, R. F. H. Hofman, and H. E. Bal. Netibis: An efficient and dynamic communication system for heterogenous grids. In *In Proc. of CCGrid 2005*, Cardiff, UK, May 2005.
- [11] T. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proc. of the Second Intl. Conference on Intelligent Systems for Molecular Biology*, pages 28–36, Menlo Park, CA, 1994. AAAI Press.
- [12] P. Barham, B. Dragovic, K. Fraser, and S. Hand et.al. Xen and the art of virtualization. In *Proc. of the 19th ACM symposium on Operating systems principles*, pages 164–177, Bolton Landing, NY, 2003.
- [13] D. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawake, and C. Packer. Beowulf: A parallel workstation for scientific computation. In *Proc. of the Intl. Conference on Parallel Processing (ICPP)*, 1995.
- [14] P. O. Boykin, J. S. A. Bridgewater, J. Kong, K. Lozev, and B. Rezaei. Brunet software library. <http://brunet.ee.ucla.edu/brunet>.
- [15] B. Calder, A. A. Chien, J. Wang, and D. Yang. The entropy virtual machine for desktop grids. In *CSE technical report CS2003-0773, University of California, San Diego*, San Diego, CA, Oct 2003.
- [16] J. Cao, O. M. K. Kwong, X. Wang, and W. Cai. A peer-to-peer approach to task scheduling in computation grid. *Intl. Journal of Grid and Utility Computing*, 1(1), 2005.
- [17] A. J. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: Self-organizing computation on a peer-to-peer network. *IEEE Transactions on Systems, Man, and Cybernetics*, 35(3), May 2005.

- [18] J. Chase, L. Grit, D. Irwin, J. Moore, and S. Sprenkle. Dynamic virtual clusters in a grid site manager. In *Proc. of the 12th Intl. Symp. on High Performance Distributed Computing*, Seattle, WA, Jun 2003.
- [19] A. S. Cheema, M. Muhammad, and I. Gupta. Peer-to-peer discovery of computational resources for grid applications. In *Proc. of the 6th IEEE/ACM Workshop on Grid Computing*, Seattle, WA, November 2005.
- [20] 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*, 33(3), 2003.
- [21] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. 2nd Symp. on Networked Systems Design and Implementation (NSDI)*, Boston, MA, May 2005.
- [22] A. Denis, O. Aumage, R. Hofman, K. Verstoep, T. Kielmann, and H. E. Bal. Wide-area communication for grids: An integrated solution to connectivity, performance and security problems. In *Proc. of the 13th Intl. Symp. on High Performance Distributed Computing*, Honolulu, Hawaii, Jun 2004.
- [23] H. E. Bal et. al. The distributed ascii supercomputer project. In *ACM Special Interest Group, Operating Systems Review*, vol34, no. 4, Oct 2000.
- [24] R. J. Figueiredo, P. Dinda, and J. A. B. Fortes. A case for grid computing on virtual machines. In *Proc. of the 23rd IEEE Intl. Conference on Distributed Computing Systems (ICDCS)*, Providence, Rhode Island, May 2003.
- [25] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. In *Proc. of the 2005 USENIX Annual Technical Conference (USENIX '05)*, Anaheim, California, Apr 2005.
- [26] J. Fortes, R. J. Figueiredo, and M. Lundstrom. Virtual computing infrastructures for nanoelectronics simulation. *Proc. of the IEEE*, 93(10):1839–1847, Aug 2005.
- [27] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proc. of the 2nd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Berkeley, CA, Feb 2003.
- [28] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Applications*, 11(2):115–128, 1997.
- [29] A. Ganguly, A. Agrawal, P. O. Boykin, and R. J. Figueiredo. IP over P2P: Enabling self-configuring virtual IP networks for grid computing. In *To appear, Proc. of the IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, Rhodes, Greece, Jun 2006.
- [30] A. S. Grimshaw and W. A. Wulf. Legion: Flexible support for wide-area computing. In *Proc. of the 7th ACM SIGOPS European Workshop*, Ireland, 1996.
- [31] S. Guha, Y. Takeda, and P. Francis. NUTSS: a SIP based approach to UDP and TCP connectivity. In *Proc. of Special Interest Group on Data Communications (SIGCOMM) Workshops, Portland, OR*, pages 43–48, Aug 2004.
- [32] A. Iamnitchi, I. Foster, and D. C. Nurmi. A peer-to-peer approach to resource location in grid environments. In *In Proc. of the 11th Symp. on High Performance Distributed Computing*, Edinburgh, UK, Aug 2002.
- [33] X. Jiang and D. Xu. Violin: Virtual internetworking on overlay infrastructure. In *Proc. of the 2nd Intl. Symp. on Parallel and Distributed Processing and Applications*, Dec 2004.
- [34] J. Kannan, A. Kubota, and K. Lakshminarayan. Supporting legacy applications over i3. Computer Science Division, University of California, Berkeley, Jun 2004. Technical report no. UCB/CSD-04-1342.
- [35] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual workspaces in the grid. In *Proc. of Europar*, Lisbon, Portugal, Sep 2005.
- [36] T. Kielmann, H. E. Bal, J. Maassen, R. van Nieuwpoort, R. Veldema, R. Hofman, C. Jacobs, and K. Verstoep. The albatross project: Parallel application support for computational grids. In *Proc. 1st European GRID Forum Workshop*, pages 341–348, Poznan, Poland, Apr 2000.
- [37] J. Kleinberg. *Nature*, 406:845, 2000.
- [38] M. Kozuch and M. Satyanarayanan. Internet suspend/resume. In *4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002), 20-21 June 2002, Callicoon, NY, USA*, pages 40–. IEEE Computer Society, 2002.
- [39] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proc. 8th IEEE Intl. Conference on Distributed Computing Systems (ICDCS)*, Jun 1988.
- [40] V. Lo, D. Zappala, D. Zhou, Y. Liu, and S. Zhao. Cluster computing on the fly: P2P scheduling of idle cycles in the internet. In *Proc. of the Third Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, CA, Feb 2004.
- [41] G. Olsen, H. Matsuda, R. Haggstrom, and R. Overbeek. fastdnaml: A tool for construction of phylogenetic trees of dna sequences using maximum likelihood. *Comput. Appl. Biosci.*, 10:41–48, 1994.
- [42] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proc. 7th IEEE Intl. Symp. on High Performance Distributed Computing (HPDC)*, Chicago, IL, Jul 1998.
- [43] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. Rfc 3489 - stun - simple traversal of user data protocol through network address translators, 2003. <http://www.ietf.org/rfc/rfc3489.txt>.
- [44] S. Santhanam, P. Elango, A. Arpaci-Dusseau, and M. Livny. Deploying virtual machines as sandboxes for the grid. In *Proc., Second USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, pages 7–12, San Francisco, CA, Dec 2005.
- [45] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proc. of USENIX Operating System Design and Implementation (OSDI)*, 2002.
- [46] J. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, 2005.
- [47] S. Son, B. Allcock, and M. Livny. Codo: Firewall traversal by cooperative on-demand opening. In *Proc. of 14th Intl. Symp. on High Performance Distributed Computing (HPDC)*, 2005.
- [48] C. Stewart, D. Hart, D. Berry, G. Olsen, E. Wernert, and W. Fischer. Parallel implementation and performance of fastdnaml - a program for maximum likelihood phylogenetic inference. In *Proc. of IEEE/ACM Supercomputing Conference (SC01)*, 2001.
- [49] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *IEEE/ACM Transactions on Networking*, vol12, no. 2, 2004 Apr.
- [50] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable Peer-to-Peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, 2003.
- [51] J. Sugerman, G. Venkitachalan, and B. H. Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proc. of the USENIX Annual Technical Conference*, Jun 2001.
- [52] A. Sundararaj and P. Dinda. Towards virtual networks for virtual machine grid computing. In *Phroc. of the 3rd USENIX Virtual Machine Research and Technology Symp.*, San Jose, CA, May 2004.
- [53] A. Sundararaj, A. Gupta, and P. Dinda. Dynamic topology adaptation of virtual networks of virtual machines. In *Proc. Seventh Workshop on Languages, Compilers and Run-time Support for Scalable Systems (LCR)*, Oct 2004.
- [54] V. Sunderam, J. Dongarra, A. Geist, and R. Manchek. The pvm concurrent computing system: Evolution, experiences, and trends. *Parallel Computing*, 20(4):531–547, Apr 1994.
- [55] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. *Beowulf Cluster Computing with Linux*, chapter Condor - A Distributed Job Scheduler. The MIT Press, 2002.
- [56] N. Therning and L. Bengtsson. Jalapeno-decentralized grid computing using peer-to-peer technology. In *In Proc. of the 2nd Conference on Computing Frontiers*, Ischia, Italy, 2005.
- [57] M. Tsugawa and J. A. B. Fortes. A virtual network (ViNe) architecture for grid computing. In *To appear, Proc. of the IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, Rhodes, Greece, Jun 2006.
- [58] X. Zhang, K. Keahey, I. Foster, and T. Freeman. Virtual cluster workspaces for grid applications. Technical Report ANL/MCS-P1246-0405, Argonne National Laboratories, Apr 2005.