Decentralized Dynamic Host Configuration in Wide-Area Overlays of Virtual Workstations

Arijit Ganguly, David Wolinsky, P. Oscar Boykin, and Renato Figueiredo
Advanced Computing and Information Systems Laboratory
University of Florida
Gainesville, FL 32611 USA
email: {aganguly, davidiw, boykin, renato}@acis.ufl.edu

Abstract

Wide-Area Overlays of Virtual Workstations (WOWs) have been shown to provide excellent infrastructure for deploying high throughput computing environments on commodity desktop machines by (1) offering scalability to a large number of nodes, (2) facilitating addition of new nodes even if they are behind NATs/Firewalls and (3) supporting unmodified applications and middleware. However, deployment of WOWs from scratch still requires setting up a bootstrapping network and managing centralized DHCP servers for IP address management. In this paper we describe novel techniques that allow multiple users to create independent, isolated virtual IP namespaces for their WOWs without requiring a dedicated bootstrapping infrastructure, and to provision dynamic host configuration (e.g. IP addresses) to unmodified DHCP clients without requiring the setup and management of a central DHCP server. We give qualitative and quantitative arguments to establish the feasibility of our approach. [1]

1. Introduction

Virtualization techniques address key challenges in the deployment of wide-area distributed computing environments. System virtual machines [36] such as VMware [38] and Xen [7] 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 [18, 26, 33]. Virtual networks ([22, 39, 25, 42]) decouple the management of address spaces and can provide full TCP/IP connectivity for nodes behind network address translation (NAT) and firewall routers. Combined, virtual machine and P2P-based self-organized virtual networking enables the deployment of scalable wide-area networks of virtual workstations (WOWs [23]). This paper describes and evaluates novel techniques that facilitate the deployment and management of WOWs by enabling 1) nodes of a WOW to dynamically obtain IP addresses using existing DHCP (Dynamic Host Configuration Protocol) clients but without relying on centralized DHCP servers, and 2) different WOWs to multiplex a single overlay network while having independently managed virtual IP address spaces.

WOWs present to end-users and applications an environment that is functionally identical to a local area network of workstations. Therefore, WOW distributed systems can be managed and programmed just like local-area networks, and can leverage unmodified subsystems such as batch schedulers, distributed file systems, and parallel application environments that are very familiar to system administrators and users. Building on scalable peer-to-peer overlay routing and discovery techniques for firewall "hole-punching", WOWs can aggregate large numbers of nodes in a virtual IP network even if they are behind NATs. Furthermore, nodes can be packaged as VM "appliances" [34, 43] that can be instantiated without disrupting the configuration of existing, commodity desktops with a variety of hosted I/O virtualization technologies (e.g. VMware, Parallels, Linux KVM). These characteristics make WOWs an excellent infrastructure for the deployment of desktop grids by

[1] 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.

supporting not only applications designed for such environments, as in [5, 9, 2, 1] and systems based on BOINC [4], but also complex, full-fledged O/S environments with unmodified software and middleware components (e.g. Condor [28, 30, 40]).

Previous work has shown that the process of adding new nodes to an existing WOW requires a simple procedure (copying and instantiating a VM image) and can be performed in a matter of seconds [23]. The bootstrapping of an initial WOW network, however, required the creation of a separate P2P overlay of (typically public) overlay nodes. Furthermore, to enable dynamic virtual IP management within a WOW as in [43], it was also required that a centralized virtual DHCP server be configured and deployed on a publically-accessible machine. In practice, the need to setup and manage the bootstrapping overlay and DHCP server can hinder the deployment of WOWs by new users. To address this problem, in this paper we present techniques that allow multiple users to create independent, isolated virtual IP namespaces for their WOWs without requiring a dedicated bootstrapping infrastructure, and to provision dynamic host configuration (e.g. IP addresses) to unmodified DHCP clients without requiring the setup and management of a central DHCP server.

To this end, we extend the IPOP virtual networking system [22] to support multiple mutually-isolated virtual networks (called IPOP namespaces) over a common P2P overlay. In particular, we propose decentralized virtual IP address management within a WOW that leverages Distributed Hash Table (DHT) functionality. Creating a new IPOP namespace only requires executing a simple program with information about the IPOP namespace (assignable virtual IP addresses and other network parameters). The namespace identifier is then provided as a parameter inside the IPOP configuration of the appliance VMs for distribution. Experiments show that a new node joining a WOW takes about 20-30 seconds on average to acquire a virtual IP address.

The rest of the paper is organized as follows. Section 2, highlights related work. In Section 3, we describe the IPOP virtual networking system that provides connectivity between WOW nodes. In Section 4, we describe our DHT implementation, which enables decentralized techniques for virtual IP address management presented in Section 5. In Section 6, we present an experimental evaluation of our approach. Section 7 concludes the paper.

2. Related work

Desktop grid environments [5, 29, 9, 2, 1] currently require tailoring the applications to handle idiosyncracies of wide-area environments with respect to host and network heterogeneity. BOINC [4] provides a platform to build applications when compute nodes are volatile desktop machines. However, in WOW the use of virtual machines and networks enables building desktop grids not only supporting these systems, but also unmodified applications and existing middleware (e.g. Globus [20], Condor [28, 30, 40]).

Our work can be classified as applying P2P techniques to computational grids and wide area clusters [19]. In [14] Cheema et.al and in [24] Iamnitchi et.al have investigated P2P discovery of computational resources for grid applications. In [10] Cao et.al. have proposed a P2P approach to task scheduling in computational grids. Related to our work are the Jalapeno [41], Organic Grid [13], OurGrid [6] and ParCop [3] 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 (we missed here on the adress management part) to enable efficient and easy-to-deploy virtualized clusters.

There is a rich literature on using P2P tecqhniques to build scalable and fault-tolerant systems. Notable among these are large scale storage utlities: CFS [16] based on Chord [37], and PAST [17] developed by Microsoft based on Pastry [32]. In [15] Cox et. al. have proposed to build Distributed DNS using DHash, a distributed hash table (DHT) based on Chord [37]. SCRIBE[12] is a large scale application-level multicast and event notification infrastructure based in Pastry P2P system. Similar to to these systems, our decentralized IP address management (1) eliminates any dedicated components, (2) scales to large numbers by harnessing the resources at participating nodes and (3) provides resilience to node failures, which are common requirements in large-scale desktop grid environments.

In [11], the authors propose to use a universal overlay to provide a scalable infrastructure to bootstrap multiple service overlays providing different functionality. It provides mechanisms to advertise services and to discover services, contact nodes, and service code. Our motivation in this work is to use a universal overlay to facilitate bootstrapping of multiple WOWs, each supporting a different community and having its own virtual private IP address space.

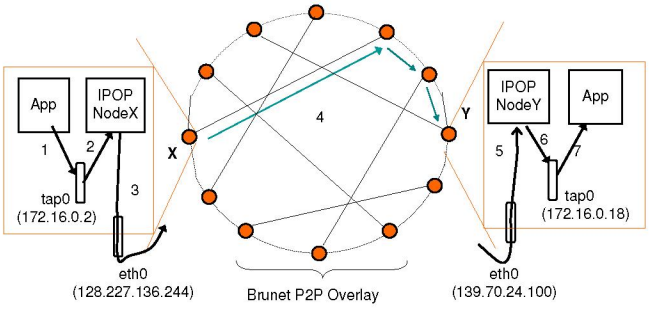

Figure 1. IPOP architecture overview.

3. IPOP - IP over P2P

In this section we describe the network virtualization technique that allows VMs within a WOW to communicate using existing TCP/IP implementations. As described in [22], IPOP runs as a user-level process inside the VM that captures packets from a virtual device *tap* inside the source VM, tunnels them over the application-level Brunet [8] P2P overlay, and then injects the packet back into a virtual device on the destination VM. It is also possible to deploy WOWs by running IPOP on the physical host and confining the cluster VMs completely within the virtual network.

Figure 1 shows the flow of data between two applications communicating over the virtual IP network provided by IPOP: 1) Application (on left) sends data to a virtual IP destination (src: 172.16.0.2, dest: 172.16.0.18). 2) IPOP reads out the ethernet frame from the *tap* and extracts the virtual IP packet, 3) The virtual IP packet is encapsulated inside a P2P (Brunet) packet addressed to P2P node Y (right) associated with the virtual IP destination, 4) and then routed within the P2P overlay to a destination node Y. 5) At node Y, IPOP extracts the virtual IP packet from the P2P packet, 6) builds an ethernet frame that it injects into the *tap*. 7) Eventually, data is delivered to application (on Y).

Earlier versions of IPOP [22, 43] have suffered from limitations with respect to:

- *Mapping from virtual IP to P2P address:* The P2P address of each IPOP instance was originally the SHA hash of the virtual IP address, which enabled nodes to quickly and independently determine the address of a P2P node on the destination VM. However, because of the one-to-one mapping from virtual IP to P2P address, a single P2P node on the host cannot route for multiple VMs inside the virtual network. When virtual IP addresses are mobile — a situation that can occur when virtual machines are allowed to migrate ([35], [39]), thus requires killing and restarting the P2P node on the target host as shown in [23].
- *Management of virtual IP addresses:* The original version of IPOP required static assignment of IP addresses. In [43], IPOP supports dynamic virtual IP configuration using unmodified DHCP clients, by capturing DHCP packets from the *tap* interface, and making requests to SOAP server that maintains virtual IP leases. With the virtual network provided by IPOP potentially involving hosts spanning wide-area networks and owned by multiple organizations, maintaining such dedicated DHCP servers is difficult. Moreover, dedicated servers introduce central points of failures.
- *Separate overlay per virtual network:* Each virtual private network of VMs must have a separate P2P overlay. This requires creating a bootstrap network of public nodes and initializing each IPOP node with the addresses of these bootstrapping nodes. This non-trivial effort hinders easy deployment of new WOWs.

In this work, we have extended the IPOP prototype to support dynamic creation and discovery of virtual IP to P2P address mappings. These mappings can be arbitrary (many-to-one), thus allowing a single P2P node to route for multiple VMs on a host. These mappings are stored as objects in the DHT. Secondly, we have extended IPOP to support different virtual private networks (each with its own address space) on top of a common P2P overlay. Each such private network is called an IPOP namespace. All nodes within a WOW node belong to the same namespace, and cannot communicate with nodes belonging to other WOWs (or namespaces).

Figure 2 shows how different WOWs (or IPOP namespaces) can exist on top of a common P2P overlay. Each virtual IP node belongs to some IPOP namespace and is associated with a P2P node. In this example, the $IP \rightarrow P2P$ mappings for nodes $A1, B1, A2, B2$ ($A1 \rightarrow X8$, $B1 \rightarrow X1$, $A2 \rightarrow X2$ and $B2 \rightarrow X4$) are stored at nodes X3, X5, X6 and X7 respectively. The DHT key for each such mapping is a combination of a globally unique identifier for the namespace and the virtual IP address within that namespace. The inclusion of the namespace identifer allows virtual IP nodes in different namespaces to have same IP addresses. To send a a virtual IP packet to node B1, the node A1 queries the DHT with $(N1, B1)$ as key. The value associated with this key is the P2P address (X1) of the P2P node associated with B1, and is quickly retrieved from node X5. From this point on communication proceeds as shown in Figure 1.

In Section 5, we describe techniques for lifecycle management of a WOW with respect to creation of an IPOP namespace and dynamic virtual IP configuration of WOW nodes. We propose a decentralized DHCP protocol to achieve dynamic network configuration that eliminates centralized components, as in [43]. We now present our DHT implementation that supports these techniques.

4. Distributed Hash Table

DHTs [37, 32, 44, 31] provide efficient object lookup by bounding the number of routing hops per lookup and also the amount of routing information per node. Each object in a DHT is associated with a key, and the ownership of keys is partitioned among participating nodes. Keys are chosen from a large space and each node is assigned an address chosen from the same space. DHTs are designed to scale to a large number of nodes and to handle continual node arrivals and failures. DHTs operate over structured P2P networks.

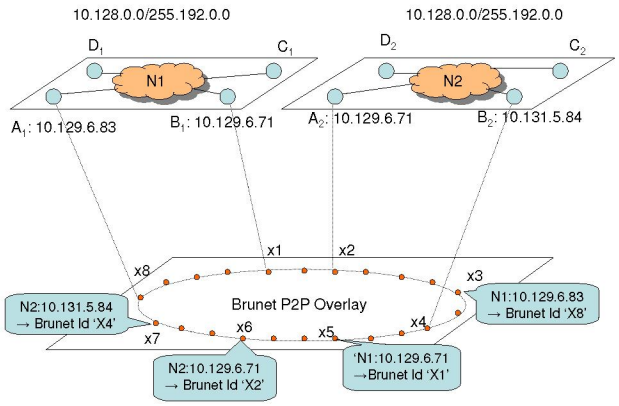

Figure 2. Example of two different WOWs with namespaces N1 and N2 sharing a common Brunet overlay.

The Brunet P2P [8] library provides mechanisms for building and maintaining structured P2P networks of overlay nodes. Brunet maintains a structured ring of P2P nodes, where each node maintains connections to its nearest neighbors in the P2P address space called *structured near* connections. Each node also maintains k connections to distant nodes in the P2P address space called *structured far* connections, Given a network of n nodes, Brunet can route a message between two nodes within $O(\frac{1}{k}\log^2(n))$ overlay hops, using the algorithm of [27].

We have extended the Brunet P2P library with functionality to support object storage and retrieval, including replication for fault-tolerance. Each DHT key is stored at two P2P nodes which are to its immediate left and right in the key (or address) space. The P2P nodes support migration of keys and their asociated values to reflect changes in the ring due to node arrival and departure. Our DHT implementation presents the following API to the applications:

1. *Create(key k, password p, value v, time-to-live ttl)*: Insert a key-value pair (k, v) into the hashtable with password p for ttl seconds, only if the key k already does not exists. Returns $true$ on success, otherwise returns an $error$. Note that the entry is stored only for ttl seconds.

2. *Delete(key k, password p)*: Delete key k and all values asociated with it, given the password p. Returns $error$ in case the key k does not exist, or the pasword does not match.

3. *ReCreate(key k, password old_p, password new_p, value new_v, time-to-live ttl)*: Equivalent to Delete(key k, password old_p) followed by Create(key k, password new_p, value new_v, time-to-live ttl).

4. *Get(key k)*: Returns all live values (*time-to-live* not expired) associated with key k.

Since the objects are stored in the DHT as soft-state for the lifetime specified in *time-to-live*, DHT aplications are thus required to re-insert (using a *ReCreate*) objects periodically into the DHT.

DHT deployments over wide-area suffer from the problem of inconsistent roots [21] because of missing overlay edges. These missing edges arise from transient Internet route outages, and the presence of NATs/Firewalls[2]. In either case, P2P nodes may have inconsistent views about the structure of the ring. This can lead to a situation where a DHT operation on a key k may not always be routed to correct nodes. It is possible that a key already exists in the DHT on certain nodes, but a subsequent *Create* operation on the same key still succeeds (instead of returning an $error$) because it is invoked on a different set of nodes which do not have the key.

To reduce the likelihood of inconsistent roots, each application specified key k is internally re-mapped to n keys $(k_1, k_2...k_n)$, which are then stored (together with the associated value) at n different locations on the P2P ring. Applications can choose this degree of re-mapping for each key, and expect DHT operations to separately provide return values for each re-mapped key. This allows applications to implement schemes like majority vote on results obtained for each such re-mapped key. For a fault to occur now, the roots of as many as half (more than one) of the re-mapped keys have to be inconsistent.

Furthermore, a new P2P node joining the overlay is not allowed to perform any DHT operation until it gets connected correctly, i.e. it forms connections with its nearest left and right neighbors on the ring. This is because an incorrectly connected node has an inconsistent view of the ring and observes roots for the DHT keys that are inconsistent with those observed by existing nodes. We have observed that it takes about 5 seconds on average for a new node to get correctlty connected.

5. WOW lifecycle management

Based on the API presented above, we now describe techniques that facilitate the deployment and management of WOWs by enabling 1) nodes of a WOW to dynamically obtain IP addresses using existing DHCP clients but without relying on centralized DHCP servers, and 2) different

[2] Although Brunet P2P library provides robust support for NAT traversal, we still do not rule out NATs which cannot be traversed

WOWs to multiplex a single overlay network while independently managing their own virtual IP address spaces.

The functionality achieved by our system is comparable with tasks an administrator would typically perform to setup a private network. Following the setting up of switches and cables, a private IP address range is set aside. Hosts connecting to the private network are assigned unique IP addresses from this IP adrress range. To enable dynamic network configuration of hosts connecting to the network, one or more DHCP servers are configured with the list of assignable IP addresses and other network parameters. New hosts discover DHCP servers through LAN broadcasts, acquire leases on IP addresses, which they renew periodically. For exchange of packets between hosts, their IP addresses must be resolved to their hardware addresses through Address Resolution Protocol (ARP). The network swicthes are automatically configured for routing packets through their ports.

In contrast, to setup a new WOW, a user is only required to create an IPOP namespace with a unique identifier and a private address space. The namespace identifier is specified inside the IPOP configuration of the WOW VM appliance image. Each deployed instance of the appliance on bootup retrieves the namespace information (virtual IP adress range) and configures itself with a unique virtual IP address. These steps are described below.

5.1. Creating an IPOP namespace

Creating an IPOP namespace is a simple procedure: executing a simple program and providing information about the namespace (assignable IP addresses, netmask and lease times). This program is already initialized with Uniform Resource Indicators (URIs, [23]) of nodes in the universal overlay and starts up a P2P node that connects to that overlay. The node tries to insert the namespace information into the DHT (using *Create*) with a randomly chosen identifier as the key. If the key already exists, the Create returns an error and the program retries with a different identifier until it succeeds. Since the DHT does not store objects indefinitely, this object holding the namespace information has to be periodically recreated (using Recreate). This namespace identifier is specified inside the IPOP configuration of the WOW appliance image.

5.2. Dynamic host configuration

In [43], IPOP supports dynamic virtual IP configuration using unmodified DHCP clients. This is achieved by capturing DHCP request packets from the *tap* and making SOAP requests to a publicly accessible server that stores the list of assignable IP addresses and active leases, and eventually injecting DHCP reponse packets to the *tap*. The SOAP server can be a single point of failure. Our decentralized DHCP uses the DHT as distributed database for storing all information the SOAP server would otherwise keep, assignable IP addresses and active leases on IP addresses.

On intercepting a DHCP packet, IPOP retrieves information about its namespace (assignable IP addresse range, netmask, lease times) from the DHT (using a *Get*) on the namespace identifer as the key. It then chooses a random IP address from that range, belonging to the namespace, and attempts to create a DHT entry (using a *Create*) with: combination of namespace identifier and the guessed IP address as the key, a randomly chosen password, and its P2P address as the value. The entry is successfully created only if there is no other entry with the same key. This prevents IP address conflicts between WOW nodes belonging to the same namespace. In case *Create* returns an error, IPOP tries another (randomly chosen) IP address until it eventually succeeds. The DHCP response packet with information about the lease is written to *tap*. The password is recorded for subsequent operations on the key.

The entry is only created with a time-to-live (TTL) equal to the lease time for that namespace, and thus needs to be recreated (using a *ReCreate*) periodically. This process is again triggered by the DHCP client, which attempts to renew a virtual IP lease after half the lease time has elapsed. In this case, IPOP attempts to *ReCreate* the same DHT key corresponding to the virtual IP address bound to *tap*.

For reasons stated earlier, our DHT implementation maps each application specified key k to 5 different keys $(k_1, k_2, k_3, k_4, k_5)$ and performs the corresponding operation on each of these keys. To consider a *Create* or *Recreate* to have successfully happened, we expect at least 3 of these operations to be successful. Otherwise, the operation is considered to have failed and a different IP address is tried.

Figure 3 shows a timeline of events, from the startup of the IPOP and DHCP client (*dhclient*) process, to having an IP address bound to *tap*. It should be noted the IP address lease acquistion cannot start until the associated P2P node is correctly connected (i.e., with left/right neighbors). Once correctly connected, IPOP tries different virtual IP addresses (one-by-one) until it is assured that no other node within the same IPOP namespace has acquired the same IP address. It is possible to have a large IP address range for the private network, which reduces the chances of two IPOP nodes guessing the same IP address.

5.3. Resolution of virtual IP to P2P address

Whenever an IPOP node has a virtual IP packet to send out, it must first must learn the P2P address of the IPOP node associated with the destination IP address. This mapping is created in the DHT when the destination node ac-

quires its virtual IP address. It can be retrieved by the source IPOP node (using *Get*) in less than a second and then cached locally. During this short period (less than a second), a few packets to that virtual IP address are dropped at the source IPOP node. As shown in [23], most TCP/IP based applications communicating over the IPOP virtual network are resilient to such transient packet losses. This process of resolving a virtual IP address to a P2P address is called "Brunet-ARP".

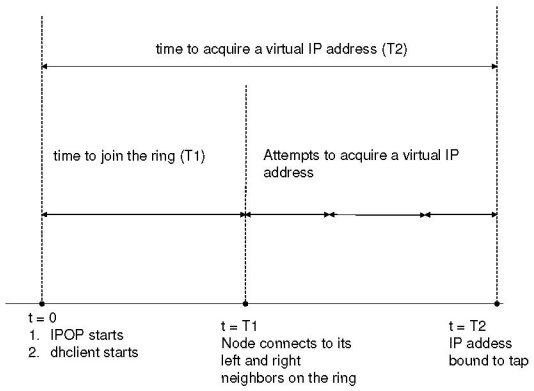

Figure 3. Events leading to virtual IP address configuration of tap interface.

6. Experiments

In this section we present an experimental evaluation of our decentralized technique for virtual IP address configuration. We measure the delay incurred from IPOP and DHCP client startup to the point that an IP address is bound to the *tap*. We set up a bootstrap overlay network of 100 P2P nodes on PlanetLab, and create an IPOP namespace with over 650000 assignable IP addresses, and and a lease time of 12 hours.

We performed an experiment between two desktop machines A and B as follows. On desktop A, we start IPOP and DHCP client so that it acquires a virtual IP address, which remains fixed during the experiment. On the desktop B, we proceeded an an iterative process of: (1) start IPOP node and DHCP client (2) wait until an IP address is bound to *tap* (3) start pinging the virtual IP address of desktop A for 200 seconds (4) kill IPOP and DHCP client on B. This process was repeated 250 times.

In each trial of the experiment, the IPOP node on desktop B had a different (randomly chosen) P2P address. The virtual IP leases from different trials persisted in the DHT (since the leases are not relinquished), and this prevented the desktop B from acquiring the same IP address over different trials. This mapping between virtual IP address and P2P address is arbirary and is discovered automatically in every trial (as described in Section 5.3) before ping packets start flowing between the desktops over the virtual network.

Figure 4 shows the cumulative distribution of delay seen by the DHCP client (*dhclient*) to acquire an IP address on the *tap*. We observe that in 90% of the cases, DHCP process finishes within 30 seconds of IPOP and DHCP client startup. As shown in Figure 3, this delay depends on (1) time taken for the IPOP nodes to get correctly connected and (2) number of differrent IP addresss that are tried. The cumulative distributions of these components are shown in Figure 5 and Figure 6.

In most cases, DHCP succeeds to acquire the first IP address it tries. However, despite picking up a large IP address range, we did observe a samples where more than one IP addresses were tried. We explain this as follows. Due to P2P message losses, the IPOP node doing DHCP did not get sufficient results to consider a *Create* or *Recreate* successful. Our implementation conservatively assumed a failure and tries a different (randomly chosen) IP address until it eventually succeeds. We are working on making our implementation more resilient to such P2P message losses.

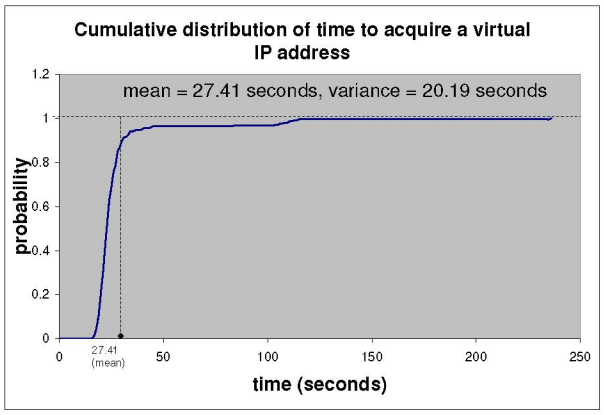

Figure 4. Time taken by a new IPOP node to acquire a virtual IP address (T2 in Figure 3).

7. Conclusions and future work

In this paper, we describe techniques that facilitate deployment of isolated WOWs by individual users without re-

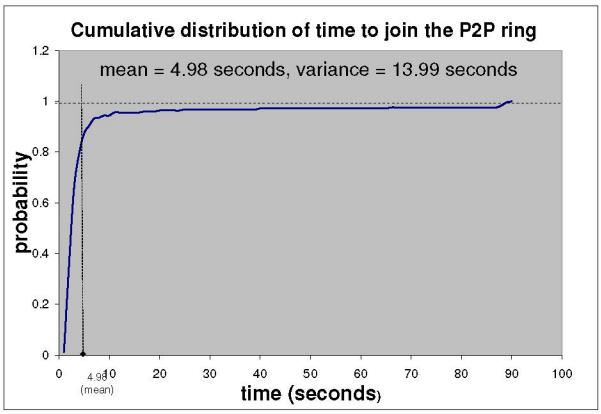

Figure 5. Time taken by a new P2P node to get connected to its left and right neighbors in the ring (T1 in Figure 3).

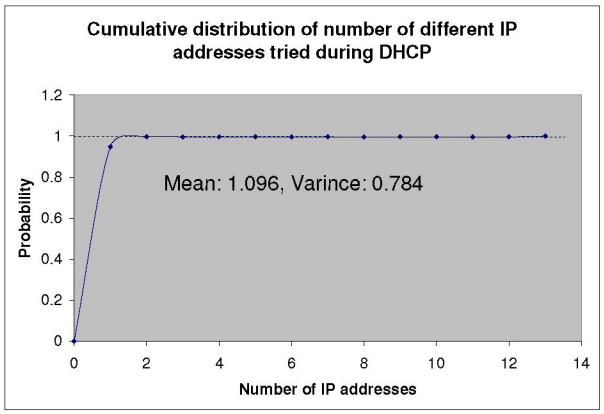

Figure 6. Number of different virtual IP addresses tried during DHCP.

quiring any bootstrapping infrastructure or centralized components. We propose a descentralized DHCP protocol for virtual IP address management within a WOW that leverages the DHT functionality of the P2P network. Experiments show that a new WOW node can acquire a unique virtual IP address withing 20-30 seconds on average.

In future work, we plan to integrate additional decentralized configuration techniques with WOW. In particular, our Condor-based WOW deployments currently are configured from a central server, and we are investigating ways to self-configure Condor pools leveraging the DHT.

References

[1] distributed.net. http://distributed.net/.

[2] fightaids@home. http://fightaidsathome.scripps.edu/.

[3] 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.

[4] D. Anderson. BOINC: A system for public-resource computing and storage. In *Proc. of the 5th Intl. Workshop on Grid Computing (GRID-2004)*, pages 4–10, Pittsburgh, PA, Nov 2004.

[5] 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.

[6] 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.

[7] P. Barham, B. Dragovic, K. Fraser, and S. H. 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.

[8] P. O. Boykin, J. S. A. Bridgewater, J. Kong, K. Lozev, and B. Rezaei. Brunet software library. `http://brunet.ee.ucla.edu/brunet`.

[9] B. Calder, A. A. Chien, J. Wang, and D. Yang. The entropia virtual machine for desktop grids. In *CSE technical report CS2003-0773, University of California, San Diego*, San Diego, CA, Oct 2003.

[10] 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.

[11] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks. In *Proc. of the SIGOPS European Workshop*, France, Sep 2002.

[12] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scalable application-level anycast for highly dynamic groups. In *Proc. of the 5th Intl. Workshop on Networked Group Communications (NGC)*, Munich, Germany, Sep 2003.

[13] 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.

[14] 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, Nov 2005.

[15] R. Cox, A. Muthitacharoen, and R. Morris. Serving dns using chord. In *Proc. of the 1st Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge,MA, Mar 2002.

[16] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cooperative file system. In *Proc. of the 18th ACM Symp. on Operating Systems Principles*, Banff, Canada, Oct 2001.

[17] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schoss Elmau, Germany, May 2001.

[18] 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.

[19] 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.

[20] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl. Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[21] M. J. Freedman, K. Lakshminarayanan, S. Rhea, , and I. Stoica. Non-transitive connectivity and dhts. In *Proc. of the 2nd USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, San Francisco, CA, Dec 2005.

[22] A. Ganguly, A. Agrawal, P. O. Boykin, and R. J. Figueiredo. IP over P2P: Enabling self-configuring virtual IP networks for grid computing. In *Proc. of the IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, Rhodes, Greece, Apr 2006.

[23] A. Ganguly, A. Agrawal, P. O. Boykin, and R. J. Figueiredo. Wow: Self-organizing wide area overlay networks of virtual workstations. In *Proc. of the IEEE Intl Symp. on High-Performance Distributed Computing (HPDC)*, Paris, France, Jun 2006.

[24] A. Iamnitchi, I. Foster, and D. C. Nurmi. A peer-to-peer approach to resource location in grid environments. In *Proc. of the 11th Symp. on High Performance Distributed Computing*, Edinburgh, UK, Aug 2002.

[25] 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.

[26] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual workspaces in the grid. In *Proc. of Europar*, Lisbon, Portugal, Sep 2005.

[27] J. Kleinberg. *Nature*, 406:845, 2000.

[28] 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.

[29] 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 3rd Intl. Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, CA, Feb 2004.

[30] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proc. of the 7th IEEE Intl. Symp. on High Performance Distributed Computing (HPDC)*, Chicago, IL, Jul 1998.

[31] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. of the ACM SIGCOMM 2001*, 2001.

[32] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of the IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, Nov 2001.

[33] S. Santhanam, P. Elango, A. Arpaci-Dusseau, and M. Livny. Deploying virtual machines as sandboxes for the grid. In *Proc. of the 2nd USENIX Workshop on Real, Large Distributed Systems (WORLDS)*, pages 7–12, San Francisco, CA, Dec 2005.

[34] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M. S. Lam, and M. Rosenblum. Virtual appliances for deploying and maintaining software. In *Proc. of the 17th Large Installation Systems Administration Conference (LISA 2003)*, October 2003.

[35] C. Sapuntzakis, B. P. R. Chandra, J. Chow, M. Lam, and M. Rosenblum. Optimizing migration of of virtual computers. *ACM SIGOPS Operating Systems Review*, 36, Issue SI:377–390, 2002.

[36] J. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, 2005.

[37] 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.

[38] 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.

[39] A. Sundararaj and P. Dinda. Towards virtual networks for virtual machine grid computing. In *Proc. of the 3rd USENIX Virtual Machine Research and Technology Symp.*, San Jose, CA, May 2004.

[40] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. *Beowulf Cluster Computing with Linux*, chapter Condor - A Distributed Job Scheduler. The MIT Press, 2002.

[41] N. Therning and L. Bengtsson. Jalapeno-decentralized grid computing using peer-to-peer technology. In *Proc. of the 2nd Conference on Computing Frontiers*, Ischia, Italy, 2005.

[42] M. Tsugawa and J. A. B. Fortes. A virtual network (ViNe) architecture for grid computing. In *Proc. of the IEEE Intl. Parallel and Distributed Processing Symp. (IPDPS)*, Rhodes, Greece, Jun 2006.

[43] D. Wolinsky, A. Agrawal, P. O. Boykin, J. Davis, A. Ganguly, V. Paramygin, P. Sheng, and R. Figueiredo. On the design of virtual machine sandboxes for distributed computing in wide area overlays of virtual workstations. In *Proc. of the 1st Workshop on Virtualization Technologies in Distributed Computing (VTDC), with Supercomputing*, Tampa, FL, Nov 2006.

[44] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE J-SAC*, 22(1):41–53, Jan 2004.