# IP Address Multiplexing for VEEs

Rayman Preet Singh
University of Waterloo
rmmathar@uwaterloo.ca

Tim Brecht
University of Waterloo
brecht@cs.uwaterloo.ca

S. Keshav
University of Waterloo
keshav@uwaterloo.ca

**Abstract** The number of publicly accessible virtual execution environments (VEEs) has been growing steadily in the past few years. To be accessible by clients, such VEEs need either a public IPv4 or a public IPv6 address. However, the pool of available public IPv4 addresses is nearly depleted and the low rate of adoption of IPv6 precludes its use. Therefore, what is needed is a way to share precious IPv4 public addresses among a large pool of VEEs. Our insight is that if an IP address is assigned at the time of a client DNS request for the VEE's name, it is possible to share a single public IP address amongst a set of VEEs whose workloads are not network intensive, such as those hosting personal servers or performing data analytics. We investigate several approaches to multiplexing a pool of global IP addresses among a large number of VEEs, and design a system that overcomes the limitations of current approaches. We perform a qualitative and quantitative comparison of these solutions. We find that upon receiving a DNS request from a client, our solution has a latency as low as 1 ms to allocate a public IP address to a VEE, while keeping the size of the required IP address pool close to the minimum possible.

## Categories and Subject Descriptors

Networks [**Network Protocols**]; Networks [**Network Architectures**]: Network design principles—*Naming and Addressing*; Networks [**Network Services**]: Naming and Addressing; Networks [**Network Services**]: Cloud Computing

## Keywords

Cloud computing, personal servers, virtual machines, IP addresses, multiplexing, IPv6, DHCP, DNS

## 1. INTRODUCTION

The number of public virtual execution environments (VEEs) is growing rapidly. For example, Amazon EC2 has maintained a steady 10% growth since 2009, surpassing 150,000 VEEs in April 2013 [1] and Rackspace has attained similar growth [6]. Each publicly-accessible VEE needs a public IPv4 or IPv6 address. Unfortunately, the IPv4 pool is nearly depleted [12] and VEE providers are reluctant to use IPv6 addresses due to the lack of widespread infrastructure to support them [2, 21, 22]. What is needed, therefore, is a way to share a single IPv4 address amongst a set of VEEs, much in the same way a NAT middlebox shares an IPv4 address amongst a set of clients.

NATs have not been used for servers in the past because, unlike clients, they are "always on". However, many applications that run on public VEEs are infrequently accessed. Moreover, clients accessing applications on different VEEs are often temporally uncorrelated. Examples include *per-user applications* [28, 29], cloud-backed mobile applications [24, 27, 36], decentralized social networks [15, 32], and participatory-sensing applications [16, 19, 20]. In such cases where many VEEs have no client requests for extended periods of time, IP addresses can be reclaimed from inactive VEEs, and allocated to a VEE that needs to serve a client request. Specifically, we can allocate an IP address to a VEE when a client resolves the VEE's name and reclaim it when the VEE is not serving any client requests, thus multiplexing a single IP address amongst a set of VEEs.

We describe several approaches to multiplexing a pool of IP addresses and find that no current approach is satisfactory. We, therefore, design and implement a simple solution that meets the design goals. In our testbed, this solution allocates IP addresses within 1 ms of a DNS request. It also keeps the IP address pool usage very close to the minimum possible.

The key contributions of this work are:

- A qualitative analysis of various approaches to IP address pool multiplexing.
- The design and implementation of a mechanism to multiplex IP addresses across VEEs using an in-VEE agent.
- A quantitative evaluation of the proposed approach and a comparison with alternative approaches.

## 2. BACKGROUND AND RELATED WORK

This section describes the background necessary for our work. We first discuss the Dynamic Host Configuration Protocol (DHCP) [11] because it is the standard way to allocate IP addresses to hosts. We also describe existing work on dynamic re-allocation of VEE resources.

### 2.1 DHCP

DHCP automates host configuration (e.g., allocation of an IP address, subnet mask, and default gateway) and can be used for the dynamic allocation of IP addresses to clients from a pool of IP addresses. It has four common message types. When a client first attempts to obtain its configuration, it broadcasts a DHCP-DISCOVER message on the physical subnet. One or more DHCP servers that are listening on that subnet then reply with a DHCP-OFFER message, which contains the client's MAC address, an IP address, the lease period and other host-specific parameters. A client accepts one of the offers with a DHCP-REQUEST;

the selected server acknowledges by sending a DHCP-ACK. The DHCP server reclaims the client's IP address when the lease period expires. A client attempts to renew the lease by sending a DHCP-REQUEST message to the server before the lease expires. Leases can then be extended or terminated by the DHCP server by sending a DHCP-ACK or DHCP-NACK, respectively.

### 2.2 Dynamic re-allocation of VEE resources:

There has been extensive work on dynamic reallocation of resources to VEEs. Wood et al. [37] propose an approach to "peek-inside" a VEE to gather OS-level statistics such as CPU, network, and memory usage, for VEE migration. Other work has used similar approaches to detect overloaded and under-loaded hosts, to perform resource control and provisioning [30], or VEE migration [23]. Inspired by this work, we propose the use of a VEE-resident daemon that monitors the VEE's network I/O and reclaims the VEE's IP address when it is no longer needed.

## 3. DESIGN GOALS

Our main design goal is to share a limited number of public IPv4 addresses among a larger set of VEEs. This high-level goal translates to the following subgoals:

- **Multi-protocol compatibility:** The solution should support various commonly-used client-server protocols, such as SMTP, POP, HTTP, SSH, IMAP, and others. Developing and maintaining separate solutions for each protocol is both expensive and inelegant.
- **Low latency:** The solution should not increase the client request response time significantly.
- **Legacy compatibility:** The solution should require no modification to existing client implementations.
- **Small IP address pool:** For a given number of VEEs, the size of the required IP address pool should be minimized.

## 4. POSSIBLE SOLUTIONS

We assume that in serving clients via their public IP addresses, VEEs have uncorrelated and low duty cycles. That is, at any instant, only a small fraction of VEEs are actively serving clients, and VEEs are unlikely to have long-standing dormant network connections. Note that VEE-hosted applications can always perform network I/O (such as download or upload data to other hosts) without a public IP address by using a private IP address on a virtual NIC. We now describe some possible solutions, (summarized in Table 1).

| | NAT | Application-level demux | DNS +DHCP | DNS +agent |
|---|---|---|---|---|
| Multi protocol | ✓ | | ✓ | ✓ |
| Low latency | ✓ | ✓ | ✓ | ✓ |
| Legacy compatibility | | ✓ | ✓ | ✓ |
| Small IP address pool | ✓ | ✓ | | ✓ |

**Table 1: Comparison of the various existing and alternative approaches for IP address multiplexing.**

### 4.1 NAT

In this approach, all VEEs share a *single* IP address and a NAT middlebox uses destination port numbers to route client requests. This approach is extremely parsimonious in its use of IP addresses, and supports different protocols, with possibly low latency. Unfortunately, it is not legacy compatible because it requires clients to use non-standard port numbers to access standard services. Due to this severe limitation we do not pursue this approach.

### 4.2 Application-level demultiplexing

In this approach, each physical host is allotted one public IP, which is shared by all its guest VEEs, and an application-specific demultiplexing daemon forwards client requests to VEEs using protocol-specific identifiers. For example, a reverse web proxy can use URL suffixes of the form http://<hostIP>/vee1 to identify the destination VEE. This solution is plausible, and, in fact, was the first one we implemented. However, it suffers from three critical problems. First, it requires developing and maintaining a demultiplexing daemon for each protocol, which is onerous. Second, when VEEs are accessed over a secure, encrypted link, such as SSH, the demultiplexer needs to play the role of a man-in-the-middle, relaying data in both directions. This can increase programming complexity significantly. A third problem is that, because the demultiplexer must map application-level users to VEEs, user names across VEEs must be unique. For example, if the user name in an SSH request is used to determine the target VEE, two VEEs cannot have a common user name (for instance, "root"). For these reasons, we do not advocate the use of this solution. However we do compare its performance with the other approaches (in Section 6).

### 4.3 DNS triggered DHCP

In this approach (shown in Figure 1), a DNS server and a DHCP server coordinate to dynamically assign IP addresses to VEEs. Each VEE is assigned a domain name that clients use to address their requests. The DNS server on the host acts as the authoritative server for the VEEs' domain names and notifies the DHCP server whenever a request for a certain VEE is received. The DHCP server chooses a lease period based on its expectation of how long it will take to service the request and reclaims the IP address when this lease period expires, denying DHCP requests to renew the lease after this time (unless there was a subsequent DNS request indicating the arrival of another client request). The DNS cache TTL is set to be the same as the lease period. This solution is discussed in more detail in Section 5.

The challenge with this approach is determining an *appropriate* lease period. A lease period that is too small may cause IP address reclamation while a VEE has an active network connection, or worse, even before the first client request arrives. VEEs may also be hosting UDP servers where client-server communication is connectionless (this issue is addressed in more detail in Section 7). On the other hand, a very large lease period may lead to IP addresses remaining allocated even when VEEs are not serving clients, thereby increasing the size of the required IP address pool. Given workload measurements, a suitable estimate of the lease period can probably be determined, but this requires extensive tuning and estimation. Thus this solution achieves multi-protocol, and legacy client compatibility but does not minimize the size of the required IP address pool, which is entirely dependent on the proper choice of the lease period.

### 4.4 DNS with in-VEE agent

Similar to the previous approach, VEEs are each assigned a domain name, which clients use to address their requests. A DNS server runs on the physical host, configured as the authoritative server for the VEEs' domain names (shown in Figure 2). It coordinates with an agent running within each VEE to dynamically assign and reclaim the IP address. When a DNS request for a VEE is received, the DNS server allocates an IP address for the VEE and commands the in-VEE agent to use that address. After the address is assigned, the agent continuously polls certain characteristics of the VEE (e.g., the number of TCP connections). It then uses them to make an informed decision to relinquish its IP address. To relinquish an address, the agent reclaims it from the interface, and notifies the DNS server, which adds it back into the pool of available IP addresses. Note that after an IP address is assigned, the agent waits for a fixed amount of time before monitoring the VEE's characteristics. This ensures that the address is not reclaimed before the first client request arrives.

This approach of overcomes the limitations of the previous approaches. It is not dependent on tuning lease periods. It supports arbitrary client-server protocols, does not limit the number of VEEs per host, requires no client modifications, and minimizes the required IP address pool size; thus achieving all our design goals. However, it relies on setting the DNS cache TTL to a small value, potentially increasing network traffic (discussed in Section 7). Note that, one of our design goals is low latency; we present a comparative evaluation in Section 6.

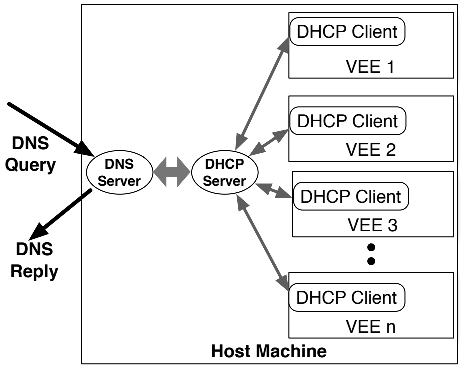

**Figure 1: IP address multiplexing using DNS triggered DHCP**

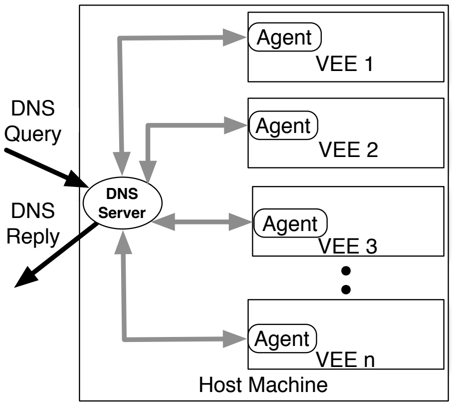

**Figure 2: IP address multiplexing using DNS with in-VEE agent**

## 5. IMPLEMENTATION

We implement three solutions: application-level request demultiplexing, DNS triggered DHCP, and DNS with in-VEE agent approach. We chose Linux Containers (LXCs), an OS-level virtualization solution for creating VEEs running Linux v3.2. OS-level virtualization solutions incur a smaller overhead than other solutions (para-virtualized or full-virtualization) [17,18,31,33–35]. Other examples of OS-level virtualization solutions are Linux vServer, OpenVZ, FreeBSD Jails, and Solaris Zones.

### 5.1 Application-level demultiplexing

We implement this approach to support VEE-hosted web applications. Each VEE is assigned a name and hosts an Apache v2.4.6 web server. Each VEE is also assigned an internal IP address using the *veth* network device [14], connected using the Linux software bridge [4] on the host. The host machine has a single NIC, which has an assigned public IP address. A proxy server receives web requests from clients, and uses the URL to determine the target VEE. Clients prefix the URL with the VEE name, e.g., /vee1/index.html. The proxy server forwards the web requests to the respective VEE using its internal IP address, and relays responses back to the clients.

### 5.2 DNS triggered DHCP

We implement, from scratch, a combined DNS and DHCP server as shown in Figure 1, which runs on the host. Each VEE runs dhclient [10], a DHCP client implementation bundled in Linux distributions like Ubuntu and Fedora.

Clients use VEEs' domain names to address their requests, which causes DNS queries, handled by the authoritative DNS server running on the host.

Figure 3 illustrates the process of IP address assignment. The VEE's DHCP client periodically broadcasts a DHCP-DISCOVER to which the DHCP server does not respond. When a DNS query is received, the DNS server allocates an available IP address to the corresponding VEE. Then the DHCP server sends a DHCP-OFFER message to the VEE, as a response to the latest DHCP-DISCOVER message. The DHCP lease period is set to a predefined value. The DHCP server confirms the allocation by sending a DHCP-ACK in response to the subsequent DHCP-REQUEST sent by the VEE. The server waits until the IP address is assigned, and sends the DNS reply to the client.

Figure 4 illustrates the process of address reclamation. At the end of the lease period (or typically 1-2 seconds prior), a VEE's DHCP client attempts to renew its IP address by sending a DHCP-REQUEST. The DHCP server responds with a DHCP-NACK and the IP address is reclaimed and added to the available address pool. This prompts the client to transmit a DHCP-DISCOVER message, which is ignored by the server, causing re-transmissions by the DHCP client, with a randomized exponential backoff (which are also ignored, as shown in Figure 3). The DHCP server stores the

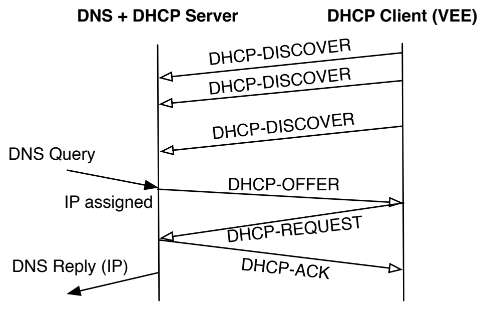

**Figure 3: IP address assignment in the DNS triggered DHCP solution.**

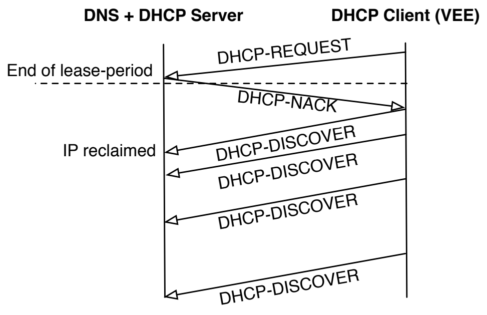

**Figure 4: IP address reclamation in the DNS triggered DHCP solution.**

transaction ID of the latest DHCP-DISCOVER message received, which is used while assigning addresses by responding with a DHCP-REQUEST.

If a DNS query for a VEE which has already been allocated an IP address is received, the lease period for that VEE is incremented. The DHCP server allows the VEE to renew its address until this updated lease period expires. Any DHCP-REQUEST messages that the VEE sends for renewal are responded to with DHCP-ACKs.

A hash table is used to maintain the mapping between a VEE's name and its MAC address, IP address (if assigned) and lease expiry timestamp. This hash table is updated whenever an address is assigned, reclaimed, or renewed. A second hash table maintains the mapping between VEEs' MAC addresses and latest DHCP-DISCOVER transaction ID. This ID is used to relate the DHCP-OFFER with the latest DHCP-DISCOVER at the time of address allocation (shown in Figure 3).

A proposed DHCP Reconfigure extension [8, 9] has recently introduced a unicast DHCP-FORCERENEW message. Using this message a server can force a DHCP client to renew its network configuration by sending a DHCP-REQUEST. In our system the DHCP server could reclaim an IP address by sending a DHCP-FORCERENEW to a VEE, and responding with a DHCP-NACK to the subsequent DHCP-REQUEST. Unfortunately, this proposed extension is absent in existing DHCP client implementations. It relies on the DHCP Authentication extension [7], which is also absent in existing DHCP client implementations. Thus, DHCP-FORCERENEW is not a viable alternative mechanism.

## 5.3 DNS with in-VEE agent

In this solution, we implement a stripped-down DNS server as shown in Figure 2, which runs on the host. An agent runs in each VEE as a daemon, to assist the host with the dynamic allocation and reclamation of IP addresses. Named pipes (one per VEE) are created on the host, and are mounted to a fixed location in the VEEs' file system using Linux bind mounts [5]. The named pipe is accessible within the VEE, which the agent reads. The DNS server sends commands to the VEE by writing them to the named pipe, and the agent reads and acts on them.

When a DNS query for a VEE is received, the DNS server allocates an available IP address to that VEE. Subsequently, it sends a "setIP" command (along with the IP address) to the VEE's agent, which configures its non-loopback interface (eth0) with that IP. The DNS server then sets the DNS cache TTL to zero and sends a DNS reply to the client. The agent then begins to monitor the VEE's network connections using the */proc* interface every 100 ms. Note that, the agent must wait for at least one VEE-client round trip time before relinquishing the IP address to ensure that the IP address is not reclaimed before the first client request arrives at the VEE. A hash table in the DNS server maintains the mapping between a VEE's name and its MAC address, IP address (if assigned) and last DNS query timestamp. The DNS server uses this hash table to answer DNS queries, and updates it whenever an address is assigned or reclaimed.

The agent observes the number of established TCP connections (under */proc/net/ip_conntrack*) every 100 ms [1], and implements a simple scheme to decide when to relinquish the IP address. The address is relinquished when after $n$ consecutive observations, the number of established TCP connections is zero. The agent performs a two-phase commit with the DNS server to withdraw the IP, and update the DNS server's hash table. This ensures consistent resolution of DNS queries while the IP address is being reclaimed.

In our current VEE and agent implementations the address assignment latency is very small, allowing us to set $n = 1$, i.e., addresses get reclaimed as soon as there are no established TCP connections. Other values of $n$ can make address reclamation less aggressive.

## 6. EVALUATION

We now evaluate and compare the three different approaches described above. VEEs are hosted on a machine with one AMD Phenom quad core 3.4 GHz processor, with 8 GB RAM. Linux Containers (LXC) [3] are used for creating the VEEs. A second machine with an Intel Core 2 Duo 2.4 GHz processor and 4 GB of RAM is used to host the clients. The two machines are connected using a 1 Gbps switch. Each VEE is assigned a static internal IP address using the *veth* [14] network device (bridged with the host's Ethernet interface using the Linux software bridge). In addition, each VEE is assigned a second interface which is used to assign and reclaim public IP addresses to the VEEs. VEE processes can perform network I/O even when the public IP address has been reclaimed, using the private IP addresses.

[1]This interval can be tuned to balance aggressiveness of reclamation and measurement overhead.

## 6.1 Microbenchmarks

We create 10 VEEs on the host. Ten clients make requests to their assigned VEEs sequentially and record the latency observed. There is a delay of 5 seconds between two client requests, i.e., at t=0, the first client makes a request to the first VEE, at t=5, the second client makes a request to the second VEE, and so forth. We use "ping" to measure round trip times, and httperf to measure the time required to download a one byte file from a VEE's web server.

**Client latency:** Table 2 shows the ping and file-download latencies for the three solutions. We compare the client latencies against those where each VEE is assigned a static public IP. We observe that the application-level demultiplexing solution incurs a delay overhead of about one millisecond for web downloads, whereas DNS-triggered DHCP incurs an overhead of about one hundred milliseconds (both for ping and file download). This is because the DHCP based solution involves two rounds of message exchanges between the server and client (DHCP-OFFER and DHCP-ACK). In contrast, DNS with in-VEE agent, has lower latency than these approaches, and is comparable to the case where VEEs have static public IP addresses.

| | **Ping** | **1 Byte download** |
|---|---|---|
| **Application-level** | NA | $2.05 \pm 0.65$ |
| **DNS+DHCP** | $109.6 \pm 3.12$ | $110.56 \pm 2.46$ |
| **DNS+agent** | $0.47 \pm 0.13$ | $1.26 \pm 0.07$ |
| **Static public IP addresses** | $0.27 \pm 0.05$ | $0.95 \pm 0.12$ |

**Table 2: Client latencies (in ms) observed in the different multiplexing solutions. All values reported with one standard deviation across 20 repetitions.**

**Size of required IP address pool:** Figure 5 shows the number of IP addresses used during the course of an experiment, where each client makes one request to one VEE. Since there is a delay of five seconds between the 10 successive client requests, the first 45 seconds result in the allocation of 10 IP addresses in the case of the DNS-triggered DHCP. As time progresses, IP addresses are reclaimed when the VEEs' lease period expires, eventually leading to all IP addresses being reclaimed. We show results for three different lease periods of 15, 30 and 60 seconds. Note that, at any instant in time during the experiment, there is at most one client request in progress. With the in-VEE agent, IP addresses get reclaimed soon after the number of established TCP connections falls to zero. Hence, in this solution there is at most one IP address in use at any instant during the experiment, matching the clients' request pattern.

In a second experiment, we configure the 10 clients to make simultaneous requests. The time period between successive requests of each client is distributed exponentially with a mean of 15 seconds. Figure 6 compares the number of IP addresses used during the experiment for the DNS-triggered DHCP, and DNS with in-VEE agent solutions. The lease period for the DHCP is set to the mean of the exponential distribution, i.e., 15 seconds. We observe that the in-VEE agent solution requires a smaller address pool than the DNS-triggered DHCP solution for the given number of VEEs. No change is observed in client latency when compared with the previous experiment.

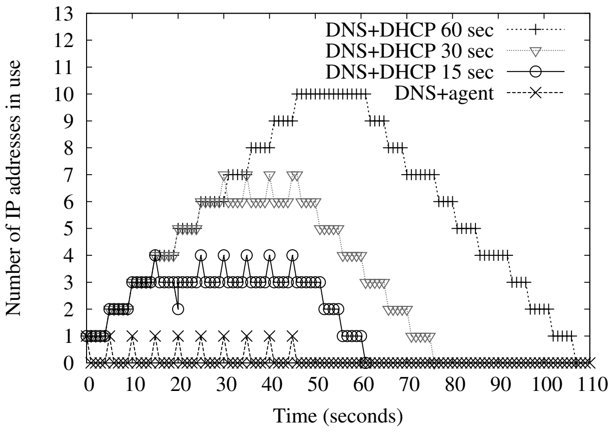

**Figure 5: Number of IP addresses in use for DNS triggered DHCP, and DNS with agent solutions.**

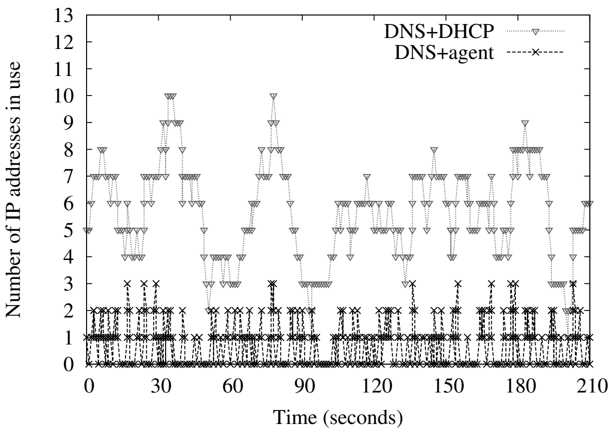

**Figure 6: Number of IP addresses in use for exponential client inter-arrival times.**

## 7. DISCUSSION

For all the three approaches that we evaluated, the latency overhead is only incurred by the first request in a client session. Thus, for all the solutions, the performance of long-lasting client-server sessions (e.g., those lasting 10 seconds or longer) is affected by less than 1%. Of the three solutions, however, only the in-VEE agent based solution minimizes the size of the required IP address pool, because it can directly monitor IP addresses that are being used by VEEs. It allows for a wide variety of address reclamation policies to be applied. In contrast, a DHCP server must somehow set the lease period to the anticipated duration for which a VEE will be serving clients. To alleviate this problem, a hybrid approach may be used, where the DHCP client coordinates with the agent, and releases the DHCP lease voluntarily if there is no network activity.

Our IP address multiplexing approach will never reclaim an IP address from a VEE as long as it has at least one active TCP connection, even if the DNS TTL associated with that connection has expired. Some clients may cache the IP address in a DNS response for use with a subsequent TCP connection request. If this happens within the TTL of the DNS response, our solution will work correctly. However, if a client were to use the cached IP address after the TTL expiry (which is deprecated), our solution may not work, because the IP address may have been reclaimed and reassigned to another VEE. In this situation, it is up to the application to reject spurious requests.

The situation is more difficult with UDP packets, in that our in-VEE agent cannot determine whether future UDP packets will be sent to this server: unlike TCP, where there is connection state, UDP has no notion of a 'connection.' In this case, our solution works correctly only if clients obey DNS TTL values when sending a UDP packet. If a client were to cache an IP address beyond the DNS TTL (a rare event, we hope), the UDP packet could be delivered to an incorrect VEE, and, as with TCP, it is up to the application to reject spurious requests. This problem is similar to the problem of using UDP to reach devices behind NAT boxes.

We have assumed that the VEEs' clients generate workloads such that only a relatively small subset of VEEs per machine are simultaneously serving clients. If this assumption is violated, it may be alleviated by load balancing requests across multiple hosts, diverting requests to the least-loaded physical server, and migrating the target VEE to that server. Existing work has demonstrated fast live-VEE migration and cloning [25,37]. Process migration solutions [13] promise the same for OS-virtualized solutions. Such work complements our approach and makes load balancing and address multiplexing possible. If the IP address pool is exhausted, it is easily detected by the DNS server. It could then delay its response in the hope that an IP address becomes available within a reasonable amount of time. Additionally, it could notify a system administrator.

Our approach results in a small increase in DNS traffic because we need to ensure that DNS responses are not cached indefinitely. It is possible to trade-off the IP address pool size for higher DNS cache TTL.

An in-VEE agent provides other possibilities. For instance, in-addition to IP address reclamation, idling VEEs can be turned off or "frozen" [26] to reclaim system resources, leading to higher level of VEE consolidation. We plan to investigate theses directions in future work.

We expect that address multiplexing will not be required once IPv6 reaches mass adoption. However, it is difficult to predict when that will occur and in the meantime this is a viable solution to the problem of addressing large numbers of VEEs that are not all simultaneously active.

## 8. CONCLUSIONS

We address the problem of multiplexing a pool of IPv4 addresses among a comparatively large number of low duty-cycle VEEs (used for hosting many applications). We evaluate possible approaches and find existing solutions to be inadequate. We design and implement an in-VEE agent based solution, which meets the design goals of incurring a low latency overhead, being legacy-compatible, and minimizing the required IP address pool size. We demonstrate the small latency overhead of our in-VEE agent-based solution, while using an off-the-shelf implementation for OS-level virtualization, and we believe that our approach is extensible to other virtualization solutions.

## 9. REFERENCES

[1] Amazon Web Services growth unrelenting. `http://news.netcraft.com/archives/2013/05/20/amazon-web-services-growth-unrelenting.html`.

[2] IPv6 adoption statistics. `http://www.google.com/ipv6/statistics.html`.

[3] Linux Containers. `http://lxc.sourceforge.net/`.

[4] Linux Ethernet Bridge. `http://http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge`.

[5] Mount option–MS_BIND. `http://man7.org/linux/man-pages/man2/mount.2.html`.

[6] Rackspace Adding 50 physical servers per day. `http://www.datacenterknowledge.com/archives/2013/08/12/rackspace-adding-servers-per-day/`.

[7] RFC 3118: Authentication for DHCP Messages. `http://tools.ietf.org/html/rfc3118`.

[8] RFC 3203: DHCP reconfigure extension. `http://http://tools.ietf.org/html/rfc3203/`.

[9] RFC 6704: Forcerenew Nonce Authentication. `http://http://tools.ietf.org/html/rfc6704/`.

[10] The Internet Systems Consortium DHCP Client. `http://linux.die.net/man/8/dhclient/`.

[11] Dynamic Host Configuration Protocol. `http://www.ietf.org/rfc/rfc2131.txt`, 1997.

[12] Free Pool of IPv4 Address Space Depleted. `http://www.nro.net/news/ipv4-free-pool-depleted`, 2011.

[13] Checkpoint/Restore In Userspace. `http://criu.org/Main_Page`, 2013.

[14] S. Bhattiprolu, E. W. Biederman, S. Hallyn, and D. Lezcano. Virtual servers and checkpoint/restart in mainstream Linux. *SIGOPS Operating Systems Review*, 2008.

[15] R. Cáceres, L. Cox, H. Lim, A. Shakimov, and A. Varshavsky. Virtual individual servers as privacy-preserving proxies for mobile devices. In *Proc. of ACM MobiHeld, 2009.*

[16] S. Chakraborty, Z. Charbiwala, H. Choi, K. R. Raghavan, and M. B. Srivastava. Fast track article: Balancing behavioral privacy and information utility in sensory data flows. *Pervasive and Mobile Computing*, 2012.

[17] V. Chaudhary, M. Cha, J. Walters, S. Guercio, and S. Gallo. A comparison of virtualization technologies for hpc. In *Advanced Information Networking and Applications (AINA), 2008.*

[18] J. Che, C. Shi, Y. Yu, and W. Lin. A Synthetical Performance Evaluation of OpenVZ, Xen and KVM. In *Proc. of IEEE APSCC 2010.*

[19] H. Choi, S. Chakraborty, Z. M. Charbiwala, and M. B. Srivastava. Sensorsafe: a framework for privacy-preserving management of personal sensory information. In *Proc. of VLDB*, 2011.

[20] D. Christin, A. Reinhardt, S. S. Kanhere, and M. Hollick. A survey on privacy in mobile participatory sensing applications. *Journal of Systems and Software*, 2011.

[21] L. Colitti, S. H. Gunderson, E. Kline, and T. Refice. Evaluating IPv6 adoption in the Internet. In *PAM*, 2010.

[22] Czyz, Jakub and Allman, Mark and Zhang, Jing and Iekel-Johnson, Scott and Osterweil, Eric and Bailey, Michael. Measuring IPv6 Adoption. Technical report, TR-13-004, ICSI, 2013.

[23] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Resource pool management: Reactive versus proactive or let's be friends. *Computer Networks*, 2009.

[24] S. Goyal and J. Carter. A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *Proc. of WMCSA, 2004.*

[25] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: rapid virtual machine

cloning for cloud computing. In *Proc. of EuroSys, 2009*.

[26] P. B. Menage. Adding Generic Process Containers to the Linux Kernel. In *Linux Symposium*. Google Inc., June 2007.

[27] E. Miluzzo, R. Cáceres, and Y.-F. Chen. Vision: mclouds-computing on clouds of mobile devices. In *Proc. of the third ACM workshop on Mobile cloud computing and services*, 2012.

[28] R. Mortier, C. Greenhalgh, D. McAuley, A. Spence, A. Madhavapeddy, J. Crowcroft, and S. Hand. The Personal Container, or Your Life in Bits. *Digital Futures*, 2010.

[29] M. Mun, S. Hao, N. Mishra, K. Shilton, J. Burke, D. Estrin, M. Hansen, and R. Govindan. Personal data vaults: a locus of control for personal data streams. In *Proc. of ACM CoNext, 2010*.

[30] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *Proc. of Eurosys '09*.

[31] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin. Performance evaluation of virtualization technologies for server consolidation. Technical report, HP, 2007.

[32] A. Shakimov, H. Lim, R. Caceres, L. Cox, K. Li, D. Liu, and A. Varshavsky. Vis-a-vis: Privacy-preserving online social networking via virtual individual servers. In *Proc. of COMSNETS, 2011*.

[33] S. Soltesz, M. Fiuczynski, L. Peterson, M. McCabe, and J. Matthews. Virtual doppelgänger: On the performance, isolation, and scalability of para-and paene-virtualized systems, 2006.

[34] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *Proc. of Eurosys*, 2007.

[35] S. Sukaridhoto, N. Funabiki, T. Nakanishi, and D. Pramadihanto. A comparative study of open source softwares for virtualization with streaming server applications. In *IEEE ISCE*, 2009.

[36] A. Wolbach, J. Harkes, S. Chellappa, and M. Satyanarayan. Transient customization of mobile computing infrastructure. In *Proceedings of MobiVirt*, 2008.

[37] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. of NSDI 2007*.