Cloud Resident Data Center

Eric Keller
Princeton University
ekeller@princeton.edu

Dmitry Drutskoy
Princeton University
drutskoy@cs.Princeton.edu

Jakub Szefer
Princeton University
szefer@princeton.edu

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

ABSTRACT

Cloud computing is transforming the way applications are created and run. The simple abstraction of leasing a virtual machine (VM) on a hosted infrastructure makes applications simple to create, rapid to expand, and economical to run. However, to gain these advantages, companies must relinquish control they have with their own private infrastructure—e.g., they cannot run their own virtualization technology or control the network elements. As a result, many companies continue to run their applications on their own private infrastructure. Rather than leasing individual VMs, we propose that customers lease an entire *cloud resident data center*—combining the control of private infrastructure with the flexibility of public infrastructure. With the cloud resident data center, a customer can design and manage the servers, network, storage, and middleboxes, as in a private infrastructure, while running in a public cloud. Our architecture presents each customer with a virtualized infrastructure, including a virtualized network topology that the customer can configure and servers that support nested virtualization. Rather than run virtualized network equipment that supports many kinds of configuration interfaces and control-plane software, our architecture uses a logically-centralized controller to emulate the control-plane functionality. We also introduce live data center migration, where we go beyond migrating individual VMs to migrate an entire data center as a single unit.

1. INTRODUCTION

The benefits of utilizing a public cloud infrastructure are well documented. In the Infrastructure as a Service (IaaS) model, the cloud provider leases virtual machines to customers, charging based on the time the VM is running. This simple abstraction is transforming the way services are run, by allowing resources to expand and contract with the needs of the service rather than provisioning for the worst case. This flexibility allows customers to establish and rapidly expand a global presence in minutes rather than days or months. Finally, the IaaS model offers a separation of expertise where the cloud provider handles all of the physical infrastructure needs (e.g., physical security).

While hosting infrastructures like Amazon EC2 provide tremendous benefits, many companies remain reluctant to move to the public cloud. Security and privacy are often touted as the main concerns, but generally speaking the issue comes down to control. To utilize the public cloud, a company must relinquish control over the software stack (e.g., running its own virtualization layer), the network (e.g., specifying quality of service parameters, placing VMs to minimize network latency, etc.), and security (e.g., controlling what applications can run on the same physical server, etc.). To retain this control, companies choose to run their applications on their own infrastructure, such as an enterprise IT infrastructure or a private data center.

To bridge the gap between public and private infrastructures, hybrid clouds have been proposed. They allow a company to simultaneously use the public cloud and private infrastructure, perhaps with a virtual private network (VPN) connecting the two [25, 2]. However, this requires companies to manage their private infrastructure *and* a second public infrastructure (perhaps using a proprietary application programming interface, API). In addition, ensuring private data stays safely in the private infrastructure requires companies to explicitly partition their data and prevent any leakage of information to the public infrastructure.

Instead, we propose the *cloud resident data center*, where, rather than leasing VMs, a company leases an entire virtual data center. The company can specify the topology of components (e.g., routers, switches, storage elements, middleboxes, and servers), configure the components, and run any software (including its own virtualization technology). The cloud resident data center runs on a shared, hosted infrastructure, enabling the customer to expand and contract the resources as needed. Additionally, this opens new possibilities for "clouds within clouds," such as creating industry and function-specific clouds without the huge cost of building a data center.

The cloud resident data center shares much with the vision of network virtualization [16] and cloud computing of today. While existing technologies (e.g., virtual network embedding [7, 26], virtualized network elements [1, 4, 17], and server virtualization) can serve as building blocks, we need to adapt, extend, and combine them to support cloud resident data centers.

In particular, customers may have many types of components (with different capabilities and configuration interfaces), including some that do not map naturally to the cloud provider's equipment—issues not addressed by prior work. To cope with this, in our architecture we (i) utilize a two layer infrastructure embedding where we first transform the 'customer virtual' topology (which may not match the available physical devices) into an 'embeddable virtual' topology (which does) and (ii) utilize a logically-central controller that emulates the collective behavior of the devices based on the customer's device-level configuration rather than virtualize each of the many different components in the network. In addition, while recent work on nested server virtualization [5] is clearly relevant, we want to reduce the amount of software common to all customers (to reduce the security threat), rather than add functionality to an already complex hypervisor. Our design incorporates nested virtualization into a "hypervisor-free" virtualization solution [19].

Additionally, we introduce *live data center migration*, where instead of migrating individual VMs, we migrate an entire data center. This simplifies the customer's move to the cloud, and also allows a company to leverage the public cloud during the early days of a new service and migrate back to a cheaper private infrastructure once the service becomes more stable. The key to supporting live data center migration is how we prepare the environment (e.g., set up the network and server containers) and migrate multiple VMs at the same time. By having a single "cut-over" point, our solution does not require temporary tunnels between the two physical infrastructures.

The remainder of the paper is organized as follows. In Section 2 we introduce the cloud resident data center. In Section 3 we describe the architecture of the cloud provider's infrastructure to enable hosting multiple cloud resident data centers. Then, in Section 4, we describe the necessary support for performing live data center migration. We conclude in Section 5.

2. CLOUD RESIDENT DATA CENTER

Rather than leasing VMs with some tacked on controls to stitch them together, we propose that the customer leases an entire *cloud resident data center*. The cloud provider offers customers the abstraction of running their own private infrastructures, while leveraging a shared, hosted infrastructure. The cloud resident data center provides control over the software stack (Section 2.1), network (Section 2.2), and security (Section 2.3) as in private infrastructures. Importantly, the infrastructure that makes this possible remains dynamic—affording customers the same ability to expand and contract as today's public cloud.

2.1 Control the Entire Software Stack

Today's public cloud (IaaS) offerings lease individual virtual machines to customers. Within these virtual machines, the customer can run any operating system and any applications. However, the customer cannot run its own native virtualization layer—the only option is for the customer to run host-based virtualization inside the leased VMs, at a tremendous hit in performance. Ironically, cloud providers make virtualization the centerpiece of their dynamic infrastructure, but do not allow their customers to take full advantage of its benefits.

Moreover, virtualization has shown great use in security, for example through the use of VM introspection [23]. Yet, customers understandably would not want the cloud provider to look into their VMs. Virtualization is also commonly used for server consolidation. In a public cloud, where there is no support for customer-initiated VM migration, the only option during slow periods is to shut down individual VM entirely (and perhaps restarting them as smaller instances).

With the cloud resident data center, customers can run any software they want—even their own virtualization technology. This enables the customer to utilize introspection, consolidation, and migration as they would in their own infrastructure.

2.2 Control the Network

Today's public clouds allow a customer to exercise limited control over how multiple virtual machines can communicate. Through provider-specific APIs the customer can establish a VPN connection to its own private infrastructure, specify access control lists, and specify which subnets are routed to the Internet gateway vs. the VPN gateway [2]. These APIs, while useful, limit the customers' control while locking them into a proprietary interface.

With intimate knowledge of its applications, the customer is in the best position to provision, place, and control network resources. The customer can manage for latency between VMs, customize the topology to match the expected communication patterns [24], control interdomain routing across multiple upstream service providers, configure access control rules to restrict access to some servers, define VLANs, and so on. For example, a cloud resident data center for a stock exchange could select paths that equalize the latency between the hosts for fair financial transactions.

With the cloud resident data center, the customer selects and configures the components in the network (routers, switches, servers, storage, middleboxes), as well as the topology between them. The configurations can be simply the configuration files used in their existing network components, rather than using a new API. Important to note is that the components the customer is using do not necessarily match what exists in the cloud provider's physical infrastructure. A customer can, for example, specify the network as a single large router [12] or switch [6].

2.3 Control the Security

In some ways, a public cloud is more secure than a customer's private infrastructure. Given the scale, the cloud provider typically has more security experts and better physical security (e.g., cameras). Additionally, cloud providers do give customers some control over the placement of the virtual machines such that different VMs can belong to different failure domains (e.g., different racks, pods, or data centers).

However, where security falls short in the public cloud is that many parties are sharing an infrastructure with a great deal of common software (e.g., the virtualization software), which a malicious party can attack to access—or obstruct access to—another's data. For example, many Amazon customers were affected by a recent outage caused by a bug Amazon's Elastic Block Storage (EBS) [3]. This is a common software layer that all customers can access, and therefore can attack. While this particular outage was not caused by a malicious attack, it very well could have been. In addition, the limited APIs for EBS meant that customers could do nothing to restore their own service. As one affected customer said, "We relied on AWS to fix the problem. Had we had more information, we would have made a different choice." [18]

With the cloud resident data center, we have a basic principle to put the customers closer to the hardware and give them more control. The job of the physical infrastructure is to isolate the customers, and let the customer build up the layers of software on top of a strong foundation. Rather than, for example, the cloud provider performing replication and load balancing for access to storage, customers see raw storage devices and utilizes their own replication software. In this way, the customer need not rely on a shared software platform.

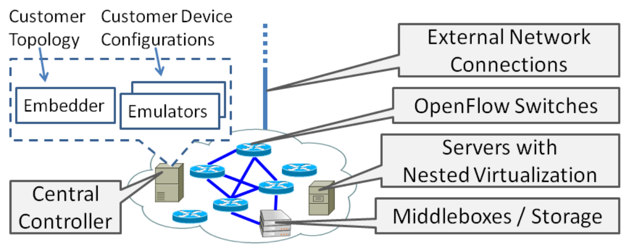
Customer Topology
Customer Device Configurations
Embedder
Emulators
External Network Connections
OpenFlow Switches
Servers with Nested Virtualization
Central Controller
Middleboxes / Storage

Figure 1: Cloud resident data center architecture.

3. ARCHITECTURE OVERVIEW

A cloud provider can support cloud resident data centers using the architecture illustrated in Figure 1. First, an embedding step maps the customer's requested resources to the underlying physical components. Then, at run time, a logically-centralized controller emulates the customer's various network components by installing the appropriate rules in a shared OpenFlow-based [15] physical network, along with some special-purpose components (e.g., storage and load balancers). For the servers, we use virtualization technology that supports nested virtualization (the ability to run a hypervisor and virtual machines inside of a virtual machine). Finally, we allow customers to communicate with the outside world—both through a VPN and the public Internet. In the rest of this section, we discuss each part of the architecture and how our solution uses, extends, and combines a variety of technologies.

3.1 Two-Level Infrastructure Embedder

The first step for creating a cloud resident data center is for the customer to specify its topology of components. Embedding software is needed to determine the availability of resources in the physical infrastructure. If resources are available, the embedding software decides which physical resources to assign to the particular customer. Virtual network embedding algorithms are not a new idea [7, 26]. However, previous work focuses on a single type of component (an abstract notion of nodes), rather than the range of devices customers run in their own networks. In addition, previous work maps each virtual node to a single physical node, which unnecessarily constrains the embedding solution. For example, a request for a 1000-port switch is more readily satisfied by distributing the virtual node over multiple physical switches.

As such, we propose a two-level embedding solution. The first step transforms the customer's request (the "customer virtual topology") into an intermediate "embeddable virtual topology", which the controller then maps on to the underlying physical network, as shown in Figure 2. The first step requires a new algorithm that considers the capabilities of the components and the topological constraints, whereas the second step can leverage existing algorithms for virtual network embedding.

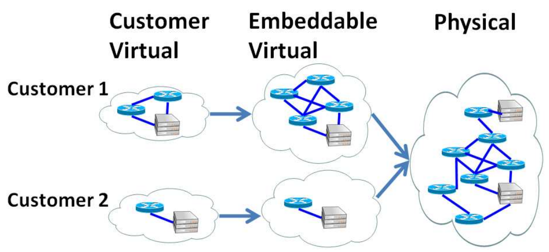

Figure 2: Two-level embedding.

The resulting mapping to the physical network is not necessarily static. The controller may recalculate the mapping under several conditions: (i) upon a request from a new customer (adjusting an existing customer may provide a better mapping for this new customer, or even enable a mapping when one is not currently possible [26]), (ii) upon a change request from customer (expanding or contracting), or (iii) upon equipment failure (remapping the affected virtual links to alternative physical paths).

3.2 Emulating Heterogeneous Components

To support multiple cloud resident data centers, we need to utilize virtualization across the cloud provider's entire underlying infrastructure. Server virtualization is well supported today through a number of commercial offerings. In addition, several solutions exist for switch and router virtualization [1, 4] where each node has multiple forwarding tables, each controlled by a different control-plane instance. However, partitioning individual network components is not a practical solution for the cloud resident data center, since it would limit customers to the specific devices (and models) in the cloud provider's physical infrastructure. This would run counter to our goal of letting customers specify and interact with network elements the same way they would in their private infrastructure.

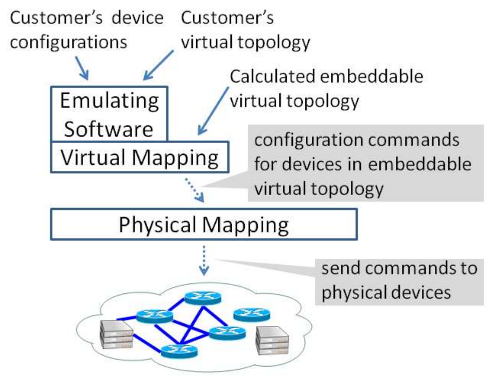
Customer's device configurations
Customer's virtual topology
Calculated embeddable virtual topology
Emulating Software
Virtual Mapping
configuration commands for devices in embeddable virtual topology
Physical Mapping
send commands to physical devices

Figure 3: Emulator.

Instead, we propose emulating virtualized network devices at the emulation software running on a logically-centralized controller, shown in Figure 3. The controller outputs commands to the underlying network with OpenFlow switches. OpenFlow switches are an appealing choice since they can forward packets along a given path, drop selected traffic, and rewrite header fields, they can act simultaneously like routers, switches, and firewalls, as well as load balancers [10, 21] and network address translators, depending on what rules the controller installs. In addition, OpenFlow switches offer a standard API to the controller [15] and naturally support data-plane virtualization by partitioning the space of packet header fields (e.g., VLAN tags, IP addresses, and so on) [17].

Still, the controller needs an effective way to translate the customer's device configurations into the low-level rules installed in the OpenFlow switches[1]. The controller could conceivably run multiple instances of control-plane software, one for each virtual node; for example, RouteFlow [14]. However, this approach introduces substantial overhead on the controller, and requires access to control-plane software from many different vendors. Instead, our controller computes the *outcome* of running a distributed control plane based on the customer's configuration and current network conditions. For example, rather than running multiple OSPF processes, we extract the OSPF configuration (e.g., link weights, area assignments, and subnet summarization) into a vendor-independent format, and perform a centralized computation (e.g., a variant of Dijkstra's algorithm) to compute the resulting shortest paths. Then, the virtual mapping software combines these results with the topology embedding information to compute the appropriate rules to install in the underlying physical switches.

Of course, OpenFlow switches are not an efficient "stand in" for all components (e.g., storage devices and some kinds of middleboxes). Instead, we support these components through special dedicated devices, virtual appliances running in a VM on the server infrastructure, or distributed implementations on end hosts [9]. For these specialized components, the controller merely performs any necessary translation between the configuration of the customer's device and the configuration of the physical devices. Storage devices are a particular challenge, since the protocols are a main differentiator between storage technologies. Rather than translating between different protocols in real time, we plan to support a selection of the most popular protocols (e.g., iSCSI and FiberChannel).

3.3 Secure Nested Server Virtualization

To manage a collection of servers as in a private infrastructure, the customer must be able to run its own virtualization technology (i.e., nested virtualization). For this, the cloud provider leases *provider VMs* to each of the customers. Within the provider VM, the customer can (if it chooses) run its own hypervisor and one or more VMs (the customer VMs). Supporting this model allows customers to perform live migration [8] of customer VMs between different provider VMs in the same cloud resident data center. This allows the customer, for example, to consolidate its customer VMs into fewer provider VMs, so it can turn off (stop paying for) some of the provider VMs.

Existing solutions for nested virtualization (e.g., Turtles [5]) add code to an already complex hypervisor—counter to our goal of to have as little software as possible directly interacting with customer code. Instead, we plan to adapt a "hypervisor-free" virtualization solution to support nested virtualization. NoHype [19] essentially supports a single level of virtualization without a hypervisor by preallocating resources and configuring the hardware such that a guest VM, slightly modified to operate within the limitations of the current implementation, can then run on bare hardware. NoHype does this by capitalizing on the virtualization support in modern hardware and the spe-

[1]An interesting special case arises if the customer's private infrastructure is already based on OpenFlow switches. In this case, the customer's "configuration" takes the form of software running on an OpenFlow controller. In this case, the cloud provider's controller can run this software and simply translate back and forth between the customer's virtual topology and the physical switches.

cialized use model in cloud computing. Rather than adding support for nested virtualization into an existing, complex hypervisor (as in Turtles), we can adapt NoHype to include a minimal hypervisor which only includes the functionality to make nested virtualization possible (e.g., to compress multiple extended page tables into one). This hypervisor is only needed until the processor supports nested virtualization.

3.4 External Internet Connections

The final aspect of the cloud resident data center architecture is connectivity to the rest of the Internet, including the customer's own enterprise users as well as the clients of its public-facing services. As in today's public cloud infrastructures, the customer can have a VPN connection back to its private facilities. In addition to private communication, we also support communication with the public Internet. Rather than simply provide a generic public Internet interface, we put the customer in greater control and involve them in the route selection process. In this case, we can utilize an existing mechanism (Transit Portal [20]) which provides each customer the illusion of having its own Border Gateway Protocol (BGP) session with the cloud provider's upstream Internet providers, though in reality these customer sessions are multiplexed over a single BGP session with each Internet provider.

4. LIVE DATA CENTER MIGRATION

The previous section assumed that the customer starts a cloud resident data center from scratch. Here, we explore *live data center migration* as a way for customers to move an entire data center without taking it off line. Migration could simplify the transition to a public cloud infrastructure, enable seamless changes between cloud providers, or allow a customer to move back to a private infrastructure. This latter is a model similar to one used by Zynga [27], where a hosted infrastructure handles the initial deployment of a game or service. Once the service becomes predictable, the company can begin using its private infrastructure.

To perform this migration we: setup the environment (Section 4.1), iteratively copy the state (Section 4.2), and finally perform a coordinated pause and resume (Section 4.3) where the final bits of state are transferred before resuming everything in the cloud provider's infrastructure.

4.1 Prepare Environment

The key to making live data center migration possible is that we are able to set up the topology within the cloud provider's infrastructure in advance, with none of the customer's software running.

The physical data center that we are migrating to will be given the customer's topology of components. The cloud provider will allocate resources based on this request. The servers that are allocated will be in the form of provider VMs. What is running in them when they are started is essentially a container that will receive the state during the next phase. For customers which have a virtualized infrastructure already, this container is simply their hypervisor with no running VMs. For customers that do not utilize virtualization, this will be an OS which supports self migration—which have been proposed [13].

The emulated elements (other than OpenFlow switches and controllers) are actually handled the same way as when the customer is creating a new cloud resident data center. Since we are simply configuring the elements and emulating their behavior, there is no state involved. The emulation software will calculate the network state based on the configuration and configure the physical components.

When the customer is running an OpenFlow network, the state in the controller is important, and therefore must be retained when migrating. Since there is a separation between the control plane (the controller) and the data plane (the switches), we can utilize a technique similar to virtual router migration [22]. Essentially, we will clone the controller to the new infrastructure and make a request for it to repopulate the data plane (the rules in the OpenFlow switches). Since the controller in the live network will still be operating while the data plane in the new network is being populated, we will need coordination between the two clones so that any new rules are consistent.

4.2 Iterative Copy of all VMs

After the environment is prepared, we essentially have VMs with nothing running that are connected by a network capable of carrying traffic (but is not). At this point, we need to begin the migration of the actual state. This includes the VMs (or servers) themselves as well as the storage on the dedicated storage devices.

Modern hypervisors come with VM migration capabilities which iteratively copies over the state while the VM is running. We can also take advantage of the significant overlap in state across different VM instances to migrate a collection of VMs with much less overhead than migrating them individually (e.g., by leveraging techniques for wide-area redundancy elimination [11]).

The process to migrate the storage state is similar. However, given that it can be orders of magnitude larger, the first copy may utilize a sneakernet, where the disks are transferred via some physical media via the mail to the cloud provider. After that, incremental differences can be obtained by examining the file access information.

4.3 Coordinated Pause and Resume

At some point, the difference between each iteration becomes small enough where further iterations will not make much progress. When this happens, we stop all of the VMs inside of the private data center being migrated. Each of the servers will copy over the remaining state, and the same will be done for the storage. They are then resumed in the new infrastructure, which has been prepared already.

This process is greatly simplified by the fact that we do not need to support a transient period where two communicating VMs can be in different locations (in which case we would need some intermediate network with tunnels to support communicate over the wide area network). Instead, we coordinate so that all migrations happen at roughly the same time and assume the applications can tolerate the small transient period when two VMs might not be able to communicate.

5. CONCLUSION

In this paper we introduced the *cloud resident data center*. To obtain both the flexibility of a public cloud infrastructure and the control available in a private infrastructure, we propose changing the abstraction of cloud computing from leasing individual virtual machines to leasing entire cloud resident data centers. Our architecture utilizes a central controller to emulate behavior at the configuration level rather than virtualize at the component level for any component that is not already well supported with virtualization (namely servers). We also utilize secure nested virtualization in order to enable the customer to control the entire software stack. Finally, we introduce live data center migration, where the entire collection of virtual machines along with the network, middleboxes, and storage elements are all migrated together.

6. REFERENCES

[1] Juniper networks: Intelligent logical router service. `http://www.juniper.net/solutions/literature/whitepapers/200097.pdf`.

[2] Amazon Virtual Private Cloud (Amazon VPC). `http://aws.amazon.com/vpc/`, 2011.

[3] Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region. `http://aws.amazon.com/message/65648/`, 2011.

[4] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI veritas: Realistic and controlled network experimentation. In *Proc. SIGCOMM*, 2006.

[5] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The Turtles project: Design and implementation of nested virtualization. In *USENIX Symposium on Operting Systems Design and Implementation (OSDI)*, 2010.

[6] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker. Virtualizing the network forwarding plane. In *Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO)*, 2010.

[7] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proc. IEEE INFOCOM*, 2009.

[8] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2005.

[9] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. ETTM: A Scalable Fault Tolerant Network Manager. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr. 2011.

[10] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. Plug-n-Serve: Load-balancing web traffic using OpenFlow, Aug. 2009. Demo at *ACM SIGCOMM*.

[11] S. Ihm, K. Park, and V. S. Pai. Wide-area network acceleration for the developing world. In *Proc. USENIX ATC*, June 2010.

[12] E. Keller and J. Rexford. The 'Platform as a Service' model for networking. In *Proc. INM/WREN*, Apr. 2010.

[13] M. A. Kozuch, M. Kaminsky, and M. P. Ryan. Migration without virtualization. In *Workshop on hot topics in Operating Systems (HotOS)*, 2009.

[14] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corra, S. C. de Lucena, and M. F. Magalhes. Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks. In *6th International Conference on Future Internet Technologies 2011 (CFI 11)*, June 2011.

[15] OpenFlow. `http://www.openflowswitch.org/`, 2011.

[16] L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet Impasse Through Virtualization. In *Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets-III)*, November 2004.

[17] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Can the production network be the test-bed? In *USENIX Symposium on Operting Systems Design and Implementation (OSDI)*, 2010.

[18] R. Stanek. Mr. Jassy, Tear Down This Wall! `http://roman.stanek.org/2011/04/21/mr-jassy-tear-down-this-wall/`, Apr. 2011.

[19] J. Szefer, E. Keller, R. B. Lee, and J. Rexford. Eliminating the hypervisor attack surface for a more secure cloud. Technical Report CE-L2011-004, Princeton University Department of Electrical Engineering, May 2011.

[20] V. Valancius, N. Feamster, J. Rexford, and A. Nakao. Wide-area route control for distributed services. In *Proc. USENIX ATC*, June 2010.

[21] R. Wang, D. Butnariu, and J. Rexford. Openflow-based server load balancing gone wild. In *Proc. Hot-ICE*, Mar. 2011.

[22] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford. Virtual routers on the move: Live router migration as a network-management primitive. In *Proc. ACM SIGCOMM*, Aug. 2008.

[23] Z. Wang, X. Jiang, W. Cui, and P. Ning. Countering kernel rootkits with lightweight hook protection. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, 2009.

[24] K. C. Webb, A. C. Snoeren, and K. Yocum. Topology switching for data center networks. In *Proc. Hot-ICE*, Mar. 2011.

[25] T. Wood, A. Gerber, K. Ramakrishnan, P. Shenoy, and J. V. der Merwe. The case for enterprise-ready virtual private clouds. In *Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2009.

[26] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM Computer Communications Review*, April 2008.

[27] Zynga. United States Security and Exchange Commission form S-1, 2011.