Scalable Network Virtualization in Software-Defined Networks

Network virtualization gives each "tenant" in a data center its own network topology and control over its traffic flow. Software-defined networking offers a standard interface between controller applications and switch-forwarding tables, and is thus a natural platform for network virtualization. Yet, supporting numerous tenants with different topologies and controller applications raises scalability challenges. The FlowN architecture gives each tenant the illusion of its own address space, topology, and controller, and leverages database technology to efficiently store and manipulate mappings between virtual networks and physical switches.

Dmitry Drutskoy
Princeton University

Eric Keller
University of Colorado

Jennifer Rexford
Princeton University

Hosted cloud computing has significantly lowered the barrier for creating new networked services. Likewise, experimental facilities such as the Global Environment for Network Innovations (GENI; www.geni.net) let researchers perform large-scale experiments on a "slice" of shared infrastructure. By letting tenants share physical resources, virtualization is a key technology in these infrastructures. Although virtual machines are now the standard abstraction for sharing computing resources, the right abstraction for networks is a subject of ongoing debate.

Existing solutions differ in the level of detail they expose to individual tenants. Amazon Elastic Compute Cloud (EC2) offers a simple abstraction in which all of a tenant's virtual machines can reach each other. Nicira extends this "one big switch" model by offering programmatic control at the network edge to enable, for example, improved access control (see http://nicira.com/en/network-virtualization-platform). Oktopus exposes a network topology so tenants can perform customized routing and access control based on knowledge about their own applications and traffic patterns.[1]

Each abstraction is most appropriate for a different class of tenants. As more companies move to the cloud, providers must go beyond network bandwidth sharing to support a wider range of abstractions. With a flexible

 Published by the IEEE Computer Society 1089-7801/13/$31.00 © 2013 IEEE

network virtualization layer, a cloud provider can support multiple abstractions ranging from a simple "one big switch" abstraction (in which tenants don't need to configure anything) to arbitrary topologies (in which tenants run their own control logic). The key to supporting various abstractions is a flexible virtualization layer that supports arbitrary topologies, address and resource isolation, and custom control logic.

Supporting numerous tenants with different abstractions raises scalability challenges. For example, supporting virtual topologies requires that tenants be able to run their own control logic and learn about relevant topology changes. *Software-defined networking* (SDN) is an appealing platform for network virtualization because each tenant's control logic can run on a controller rather than on physical switches. In particular, OpenFlow offers a standard API for installing packet-forwarding rules, querying traffic statistics, and learning about topology changes.[2] Supporting multiple virtual networks with different topologies requires a way to map a rule or query issued on a virtual network to the corresponding physical switches, and to map a physical event (such as a link or switch failure) to the affected virtual components. Any virtualization solution must perform these mapping operations quickly, to give each tenant real-time control over its virtual network.

Here, we present FlowN, an efficient and scalable virtualization solution. FlowN is built on top of SDN technology to provide programmable control over a network of switches. Tenants can specify their own address space, topology, and control logic. The FlowN architecture leverages advances in database technology to scalably map between the virtual and physical networks. Similarly, FlowN uses a shared controller platform, analogous to container-based virtualization, to efficiently run tenants' controller applications. Experiments with our prototype FlowN system, built as an extension to the NOX[3] OpenFlow controller, show that these two design decisions lead to a fast, flexible, and scalable solution for network virtualization.

Network Virtualization

For hosted and shared infrastructures, as with cloud computing, providers should fully virtualize an SDN to represent the network to tenants. We next look at the requirements for virtualization in terms of specifying and isolating virtual infrastructures.

SDN Controller Application

To support the widest variety of tenants, a cloud provider should let each one specify custom control logic on its own network topology. SDN is quickly gaining traction as a way to program the network. In SDN, a logically centralized controller manages the collection of switches through a standard interface, letting the software control switches from different vendors. With the OpenFlow standard,[2] for example, the controller's interface to a hardware switch is effectively a flow table with a prioritized list of rules. Each rule encompasses a pattern that matches bits of the incoming packets, and actions that specify how to handle these packets. These actions include forwarding out of a specific port, dropping the packet, or sending the packet to the controller for further processing. The software controller interacts with the switches (for instance, handling packets sent to the controller) and installs the flow-table entries (for example, installing rules in a series of switches to establish a path between two hosts).

With FlowN, each tenant can run its own controller application. Of course, not all tenants need this much control. Tenants wanting a simpler network representation can simply choose from default controller applications, such as all-to-all connectivity (similar to what Amazon EC2 offers) or an interface similar to a router (as with RouteFlow[4]). This default controller application would run on top of the virtualization layer that FlowN provides. As such, tenants can decide whether they want full network control or a preexisting abstraction that matches their needs.

Virtual Network Topology

In addition to running a controller application, tenants also specify a network topology. This lets each tenant design a network for its own needs, such as favoring low latency for high-performance computing workloads or a high bisection bandwidth for data processing workloads.[5] With FlowN, each virtual topology consists of nodes, interfaces, and links. Virtual nodes can be either a server (virtual machine) or an SDN-based switch. Each node has a set of virtual interfaces that connect to other virtual interfaces via virtual links. Each virtual

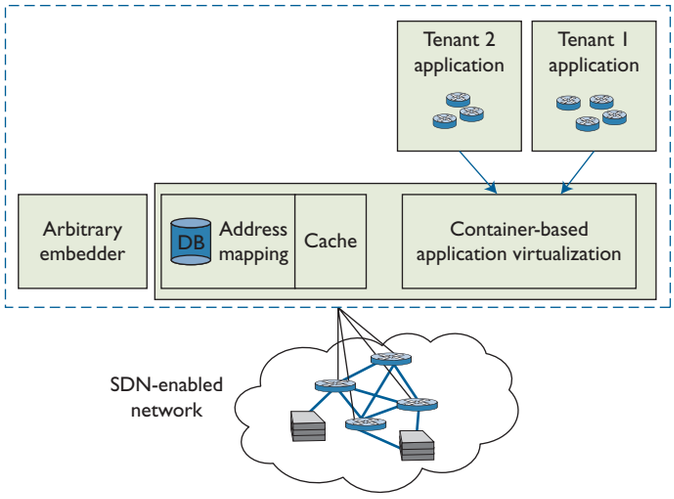

Figure 1. System design. The FlowN architecture lets tenants write arbitrary controller software to fully control their own address space, defined by the fields in the packet header, and uses modern database technology to perform the mapping between the virtual and physical address space.

component can include resource constraints — for example, the maximum number of flow-table entries on the switch, the number of cores on a server, or the bandwidth and maximum latency for virtual links. The cloud provider runs an embedding algorithm[6] to map the requested virtual resources to the available physical ones.

Importantly, with full virtualization, virtual topologies are decoupled from the physical infrastructure. This is in contrast to "slicing" the physical resources (as occurs with FlowVisor[7]), which lets tenants run their own controller over a portion of the traffic and a subset of the physical network. However, with slicing, the mapping between virtual and physical topologies is visible to the tenants. With FlowN, mappings aren't exposed; instead, tenants simply see their virtual networks. With this decoupling, cloud providers can offer virtual topologies with richer connectivity than the physical network, or remap virtual networks to hide the effects of failures or planned maintenance. Virtual nodes, whether switches or virtual machines, can move to different physical nodes without changing the tenant's network view.

Address Space and Bandwidth Isolation

Each tenant has an address space, defined by fields in the packet headers (source and destination IP address, TCP port numbers, and so on). Rather than divide the available address space among tenants, FlowN present virtual address spaces to each application. This gives each tenant control over all fields within the header (for instance, two tenants can use the same private IP addresses). To do this, the FlowN virtualization layer maps between the virtual and physical addresses. To distinguish between the traffic and rules for different tenants, the edge switches encapsulate incoming packets with a protocol-agnostic extra header, transparent to the tenant's virtual machines and controller application. This extra header (for example, a virtual LAN, or VLAN) simply identifies the tenant — we don't run the associated protocol logic (for example, per the VLAN spanning-tree protocol).

In addition to address-space isolation, the virtualization solution must support bandwidth isolation. Although current SDN hardware doesn't let network controllers limit bandwidth usage, the recent OpenFlow specification includes this capability.[8] Using embedding algorithms, we guarantee bandwidth to each virtual link. As support for enforcing these allocations becomes available, we can incorporate them into FlowN.

FlowN Architecture Overview

Hosted cloud infrastructures are typically large datacenters that have many tenants. As such, our virtualization solution must scale in both the physical network's size and the number of virtual networks. Being scalable and efficient is especially critical in SDNs, where packets are not only handled in the hardware switches, but can also be sent to the centralized controller for processing.

Virtualization has two main performance issues in an SDN context. First, an SDN controller must interact with switches through a reliable communication channel (such as SSL over TCP) and maintain a current view of the physical infrastructure (for example, which switches are alive). This incurs both memory and processing overhead, and introduces latency. Second, with virtualization, any interaction between a tenant's controller application and the physical switches must go through a mapping between the virtual and physical networks. As the number of virtual and physical switches increases, performing this mapping becomes a limiting factor in scalability.

To overcome these issues, the FlowN architecture (see Figure 1) is based around two key

design decisions. First, FlowN lets tenants write arbitrary controller software that has full control over the address space and can target an arbitrary virtual topology. However, we use a shared controller platform (such as NOX[3]) rather than running a separate controller for each tenant. This approach is analogous to container-based virtualization such as LXC for Linux (http://lxc.sourceforge.net) or FreeBSD Jails (www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html). Second, we use modern database technology to perform the mapping between the virtual and physical address space. This provides a scalable solution that's easily extensible as new functionality is needed.

Container-Based Virtualization

Each tenant has a controller application that runs on top of its virtual topology. This application consists of handlers that respond to network events (such as topology changes, packet arrivals, and new traffic statistics) by sending new commands to the underlying switches. Each application should have the illusion **//that it?//** runs on its own controller. However, running a full-fledged controller for each tenant is unnecessarily expensive. Instead, FlowN supports container-based virtualization by mapping API calls in the NOX controller back and forth between the physical and virtual networks.

Full Controller Virtualization Overhead

Running a separate controller for each tenant seems like a natural way to support network virtualization. In this solution, the virtualization system exchanges OpenFlow messages directly with the underlying switches, and with each tenant's controller. This system maintains the relationships between physical and virtual components, and whatever encapsulation is applied to each tenant's traffic. When physical events happen in the network (for example, a virtual link or switch failure, or a packet-in event for a particular virtual network), the system translates them to one or more virtual events and sends the corresponding OpenFlow message to the appropriate tenants. Similarly, when a tenant's controller sends an OpenFlow message, the virtualization system converts the message (for example, by mapping virtual switch identifiers to physical switch identifiers, including the tenant-specific encapsulation header in the packet-handling rules) before sending a message to the physical switch.

The FlowVisor system follows this approach,[7] virtualizing the switch data plane by mapping OpenFlow messages sent between the switches and the per-tenant controllers. Using the OpenFlow standard as the interface to the virtualization system has some advantages (tenants can select any controller platform, for instance), but introduces unnecessary overhead. Repeatedly marshalling and unmarshalling parameters in OpenFlow messages incurs extra latency. Running a complete controller instance for each tenant involves running a large code base, which consumes extra memory. Periodically checking for the separate controllers' "liveness" incurs additional overhead. The overhead for supporting a single tenant might not be that significant. However, when we consider that the virtualization layer will now have to provide the full interface of switches for each virtual switch (which will outnumber the physical switches by at least an order of magnitude), the cumulative overhead is significant — requiring more computing resources and incurring extra, unnecessary latency.

Container-Based Controller Virtualization

Instead, we adopt a solution inspired by container-based virtualization, in which a shared kernel runs multiple user-space containers with independent name spaces and resource scheduling. FlowN is a modified NOX controller that can run multiple applications, each with its own address space, virtual topology, and event handlers. Rather than map OpenFlow protocol messages, FlowN maps between NOX API calls. In essence, FlowN is a special NOX application that runs its own event handlers that call tenant-specific event handlers. For example, when a packet arrives at the controller, FlowN runs its packet-in event handler, identifying the appropriate tenant (based on the VLAN tag on the packet, for example) and invoking that tenant's own packet-in handler. Similarly, if a physical port fails, FlowN's port-status event handler identifies the virtual links traversing the failed physical port, and invokes the port-status event handler for each affected tenant with the ID of its failed virtual port.

Similarly, when a tenant's event handler invokes an API call, FlowN intercepts the call and translates between the virtual and physical components. Suppose a tenant calls a

function that installs a packet-handling rule in a switch. FlowN maps the virtual switch ID to the corresponding physical switch's identifier, checks that the tenant hasn't exceeded its allotted space for rules on that switch, and modifies the pattern and actions in the rule. When modifying a rule, FlowN changes the pattern to include the tenant-specific VLAN tag, and the actions to forward on to the physical ports associated with the tenant's virtual ports. Next, FlowN invokes the underlying NOX function to install the modified rule in the associated physical switch. FlowN follows a similar approach to intercept other API calls for removing rules, querying traffic statistics, sending packets, and so on.

Each tenant's event handlers run within its own thread. Although we haven't incorporated any strict resource limits, CPU scheduling provides fairness among the threads. Furthermore, running a separate thread per tenant protects against a tenant's controller application not returning (for example, having an infinite loop) and preventing other controller applications from running.

Database-Driven Mappings

Container-based controller virtualization reduces the overhead of running multiple controller applications. However, any interaction between a virtual topology and the physical network still requires a mapping between virtual and physical spaces. This can easily become a bottleneck, which FlowN overcomes by leveraging advances in database technology.

Virtual Network Mapping Overhead

To provide each tenant with its own address space and topology, FlowN maps between virtual and physical resources. It encapsulates tenants' packets with a unique header field (for example, a VLAN tag) as they enter the network. To support numerous tenants, the switches swap the labels at each hop in the network. This lets a switch classify packets based on the physical interface port, the label in the encapsulation header, and the fields the tenant application has specified. Each switch can thus uniquely determine which actions to perform.

The virtualization software running on the controller determines these labels. A virtual-to-physical mapping occurs when an application modifies the flow table (adds a new flow rule, for example). The virtualization layer must alter the rules to uniquely identify the virtual link or switch. A physical-to-virtual mapping occurs when the physical switch sends a message to the controller (as when a packet doesn't match any flow-table rule). The virtualization layer must de-multiplex the packet to the right tenant, and identify the right virtual port and virtual switch.

These mappings can occur either one-to-one, as when installing a new rule or handling a packet sent to the controller, or one-to-many, as with link failures that affect multiple tenants. In general, these mappings are based on various combinations of input and output parameters. Using a custom data structure with custom code to perform these mappings can easily become unwieldy, leading to software that's difficult to maintain and extend.

More importantly, this custom software would need to scale across multiple physical controllers. Depending on the mappings' complexity, a single controller machine eventually hits a limit on how many mappings it can perform per second. To scale further, the controller can run on multiple physical servers. With custom code and in-memory data structures, distributing the state and logic in a consistent fashion becomes extremely difficult.

FlowVisor takes this custom data-structure approach.[7] Although FlowVisor doesn't provide full virtualization — it instead slices the network resources — it must still map an incoming packet to the appropriate slice. In some cases, hashing can help perform a fast lookup, but this isn't always possible. For example, for the one-to-many physical to virtual mappings, FlowVisor iterates over all tenants, and for each tenant, it performs a lookup with the physical identifier.

Topology Mapping with a Database

Rather than using an in-memory data structure with custom mapping code, FlowN uses modern database technology. Both the topology descriptions and the assignment to physical resources lend themselves directly to a database-style relational model. Each virtual topology is uniquely identified by some key, and consists of several nodes, interfaces, and links. Nodes contain the corresponding interfaces, and links connect one interface to another. FlowN describes physical topology similarly. It maps each virtual node to one physical node; each virtual link

becomes a path, or a collection of physical links and a hop counter giving those links' ordering.

FlowN stores mapping information in two tables. The first stores the node assignments, mapping each virtual node to one physical node. The second stores the path assignment by mapping each virtual link to a set of physical links, each with a hop count number that increases in the path direction.

Mapping between virtual and physical space then becomes a simple matter of issuing an SQL query. For example, for packets received at the controller for software processing, we must remove the encapsulation tag and modify the identifier for the switch and port on which the packet was received. We can realize this with the following query:

```
SELECT L.Tenant_ID, L.node_ID1,
  L.node_port1
   FROM Tenant_Link L, Node_T2P_Mapping M
   WHERE VLAN_tag = x AND
     M.physical_node_ID = y
         AND M.tenant_ID = L.tenant_ID
         AND L.node_ID1 = M.tenant_node_ID
```

Other events are handled similarly, including lookups that yield multiple results — for instance, when a physical switch fails, and we must fail all virtual switches currently mapped to that physical switch.

Although using a relational database reduces code complexity, the real advantage is that we can capitalize on years of research to achieve a durable state and highly scalable system. Because we expect many more reads than writes in this database, we can run a master database server that handles any writes (for topology changes and adding new virtual networks, for example). FlowN then uses multiple slave servers to replicate the state across multiple servers. Because the mappings don't change often, the system can then use caching to optimize for mappings that frequently occur.

With a replicated database, we can partition the FlowN virtualization layer across multiple physical servers, colocated with each replica of the database. Each physical server interfaces with a subset of the physical switches and performs the necessary physical-to-virtual mappings. These servers also run the controller application for a subset of tenants and perform the associated virtual-to-physical mappings. In some cases, a tenant's virtual network might span physical switches that different controller servers have handled. In this case, FlowN simply sends a message from one controller server (say, responsible for the physical switch) to another (running the tenant's controller application) over a TCP connection. More efficient algorithms for assigning tenants and switches to servers are an interesting area for future research.

Evaluation

FlowN is a scalable and efficient SDN virtualization system. Here, we compare our FlowN prototype with unvirtualized NOX — to determine the virtualization layer overhead — and FlowVisor — to evaluate scalability and efficiency.

We built a FlowN prototype by extending the Python NOX version 1.0 OpenFlow controller.[3] This controller runs without any applications initially, instead providing an interface to add applications (that is, for each tenant) at runtime. Our algorithm for embedding new virtual networks is based on related work.[6] The embedder populates a MySQL version 14.14 database with mappings between virtual and physical spaces. We implement all schemes using the InnoDB engine. For encapsulation, we use the VLAN header, pushing a new header at the ingress of the physical network, swapping labels as necessary in the core, and popping the header at the egress. Alongside each database replica, we run a memcached instance that gets populated with the database lookup results and provides faster access times should a lookup be repeated.

We run our prototype on a virtual machine running Ubuntu 10.04 LTS, given full resources from three processors of a i5-2500 CPU at 3.30 GHz, 2 Gbytes of memory, and an SSD drive (Crucial m4 SSD 64 Gbyte). We perform tests by simulating OpenFlow network operation on another virtual machine (running on an isolated processor with its own memory space) using a modified cbench[9] to generate packets with the correct encapsulation tags.

We then measure latency by measuring the time between when cbench generates a packet-in event and when it receives a response. The virtual network controllers for each network are simple learning switches that operate on individual switches. In our setup, each new packet-in event triggers a rule installation, which the cbench application receives.

Related Work in Network Virtualization

Researchers have proposed network virtualization in various contexts. In early work, ATM switches were partitioned into "switchlets" to enable the dynamic creation of virtual networks.[1] In industry, router virtualization is already available in commercial routers (www.juniper.net/techpubs/software/erx/junose80/swconfig-system-basics/html/virtual-router-config.html). This lets multiple service providers share the same physical infrastructure, as with the Virtual Network Infrastructure (VINI),[2] or as a means for a single provider to simplify management of a single physical infrastructure among many services, as with ShadowNet.[3]

More recently, researchers have introduced network virtualization solutions in the context of software-defined networks (SDNs) to complement the virtualized computing infrastructure in multitenant datacenters (see www.necam.com/PFlow/doc.cfm?t=PFlowController or http://nicira.com/en/network-virtualization-platform).[4] Although considerable work has been done for network virtualization in the SDN environment, current solutions differ from the approach we describe in the main text in how they split the address space and represent virtual topology. With FlowN we fully virtualize the address space and the topology, with a scalable and efficient system.

References

1. J. van der Merwe and I. Leslie, "Switchlets and Dynamic Virtual ATM Networks," *Proc. IFIP/IEEE Int'l Symp. Integrated Network Management*, IEEE, 1997, pp. 355–378.
2. A. Bavier et al., "In VINI Veritas: Realistic and Controlled Network Experimentation," *Proc. ACM SIGCOMM 2006 Conf.*, ACM, 2006, pp. 3–14.
3. X. Chen, Z.M. Mao, and J. van der Merwe, "ShadowNet: A Platform for Rapid and Safe Network Evolution," *Proc. Usenix Ann. Technical Conf.*, Usenix Assoc., 2009, pp. 29–42.
4. R. Sherwood et al., "Can the Production Network Be the Testbed?" *Proc. 9th Conf. Operating Systems Design and Implementation*, Usenix Assoc., 2010, article no. 1–6.

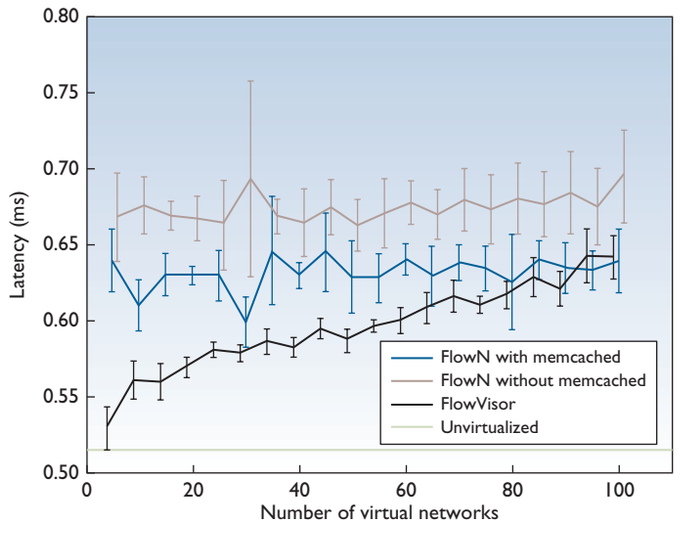

Figure 2. Latency versus virtual network count. As the number of virtual networks the virtualization layer must support increases, the overhead, as measured by the latency in how long it takes a control message to be processed, also increases. As seen here, FlowN has a higher overhead due to the database, but scales better than FlowVisor.

Although directly comparing FlowN and FlowVisor is difficult because they're performing different functionality (see Figure 2), FlowN has a much slower increase in latency than FlowVisor as more virtual networks are added. FlowVisor has lower latency for small numbers of virtual networks as the overhead of using a database, as compared to a custom data structure, dominates at these small scales. However, at larger sizes (roughly 100 virtual networks and greater), the scalability of the database approach that FlowN uses wins out. The overall increase in latency over the unvirtualized case is less than 0.2 ms for the prototype that uses memcached and 0.3 ms for the one that doesn't.

As we move toward virtualized and, in many cases, multitenant computing infrastructures, the network must also be virtualized to complement the already virtualized servers. SDNs give us the tools to achieve this virtualization, but to date, many have taken a stance on what interface, or abstraction, should be provided to the network operators. Rather than take a stand on what abstraction users of these virtualized infrastructures want, we instead provide a flexible system for partitioning resources with FlowN (on which support for different abstractions can be built). Importantly, the system's scalability is paramount. With FlowN, we capitalize on modern database technology and use ideas from container-based virtualization to allow the network virtualization layer to scale beyond the limits of today's virtualization technology. In doing so, we move a step closer toward cloud computing infrastructures in which tenants are in greater control over their own networks.

References

1. C. Wilson et al., "Better Never than Late: Meeting Deadlines in Datacenter Networks," *Proc. ACM SIGCOMM 2011 Conf.*, ACM, 2011, pp. 50–61.
2. N. McKeown et al., "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Comm. Rev.*, vol. 38, no. 2, 2008, pp. 69–74.
3. N. Gude et al., "NOX: Towards an Operating System for Networks," *ACM SIGCOMM Computer Communication Rev.*, vol. 38, no. 3, 2008, pp. 105–110.
4. M.R. Nascimento et al., "Virtual Routers as a Service: The RouteFlow Approach Leveraging Software-Defined Networks," *Proc. Conf. Future Internet Technologies* (CFI), ACM, 2011, pp. 34–37.
5. K. Webb, A. Snoeren, and K. Yocum, "Topology Switching for Data Center Networks," *Proc 11th Usenix Conf. Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services* (Hot-ICE), Usenix Assoc., 2011.
6. N. Chowdhury, M. Rahman, and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," *Proc. 28th IEEE Conf. Computer Communications* (INFOCOM 09), IEEE, 2009, pp. 783–791.
7. R. Sherwood et al., "Can the Production Network Be the Testbed?" *Operating Systems Design and Implementation*, Oct. 2010, pp. 365–378.
8. Openflow Switch Specification 1.3.0., Open Networking Foundation, Apr. 2012; www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf.
9. *Openflow Operations Per Second Controller Benchmark*, OpenFlow specification, Mar. 2011; www.openflow.org/wk/index.php/Oflops.

Dmitry Drutskoy is a computer scientist at Elysium Digital, an IP litigation consulting and digital forensics company in Boston. His research interests are in computer networks, databases, and 3D graphics. Drutskoy has an MSE in computer science from Princeton University. Contact him at drutskoy@cs.princeton.edu.

Eric Keller is an assistant professor at the University of Colorado. His overall research aim is to create a secure and reliable end-to-end infrastructure for dependable networked services, using a cross-layer approach from networking, computer architecture, operating systems, and distributed systems. Keller has a PhD in electrical engineering from Princeton University. Contact him at eric.keller@colorado.edu.

Jennifer Rexford is a professor in the computer science department at Princeton University. Her research interests include Internet routing, network measurement, and network management, with the larger goal of making data networks easier to design, understand, and manage. Rexford has a PhD in computer science and electrical engineering from the University of Michigan. She's the coauthor of Web Protocols and Practice (Addison-Wesley, 2001), and a member of IEEE. Contact her at jrex@cs.princeton.edu.