Safely Using Your Production Network as a Testbed

ROB SHERWOOD

Rob Sherwood is a Senior Research Scientist at Deutsche Telekom Inc.'s R&D Lab in Los Altos, California. Additionally, Rob is a member of the Clean Slate Lab at Stanford University.
r.sherwood@telekom.com

FlowVisor is a prototype implementation of *network slicing*, a technique for allowing production and experimental network protocols to safely share the same physical infrastructure. FlowVisor relies on OpenFlow, a new protocol for managing switches and routers that controls network traffic using patterns found in packets.

Network administrators must strike a careful balance between providing a solid, reliable network and a network that has the cutting-edge services demanded by its users (e.g., multicast, IPv6, IP mobility, or something more radical such as a traffic load balancing service). This is particularly challenging in a research or university setting, as some of the requested services are themselves experimental and therefore already disruption-prone in nature. Further complicating this issue are up-and-coming technologies like OpenFlow [3] that allow network administrators, operators, and researchers to program custom services into the network. OpenFlow provides programmatic control of how packets are forwarded, and with it researchers are prototyping novel network services [1, 2]. But while OpenFlow allows interesting new network services, this additional freedom comes at the cost of additional potential sources of network instability.

Common practice is to deploy new services in a smaller testbed or isolated VLAN and then, after some time has passed and confidence has been built, transition the service to the production network. This approach has two main shortcomings. First, typically for reasons of cost, testbeds rarely have the same size, number of users, heterogeneity, or, more generally, complexity as the real production networks. As a result, it's not uncommon for a service to work correctly in the testbed but still have problems in the production network. Second, once the service has passed its testbed evaluation, there is typically not an incremental and controlled (e.g., user-by-user) way of deploying the service in the production network. For example, on most routers, multicast support is a binary feature: it is either enabled or disabled, so some error in the service could affect all users, not just the ones that elect to use multicast.

Our approach, as demonstrated by our first prototype, called FlowVisor [7], is to divide the network into logical partitions, called *slices*, and to give each service control over packet forwarding in its own network slice. Users then *opt in* to one or more network services: that is, they delegate control of subsets of their traffic to specific services. The existing production network services (e.g., Spanning Tree, OSPF, etc.) run in a slice, and by default, all users are opted into the production slice. Critically, the FlowVisor ensures strong isolation between slices: that is, actions in one slice do not affect another slice.

For example, a network administrator, Alice, could use FlowVisor to divide her network into three slices (Figure 1). The first slice the administrator keeps for herself as a *production* slice and in it runs standard, well-vetted network protocols, e.g., OSPF, Spanning Tree, or basic MAC-address learning/switching using available open source software such as Quagga [6] or NOX [4]. The administrator then delegates the second and third slices to two network researchers, Bob and Cathy. Bob is developing a network service optimized for high throughput, and Cathy is running a service optimized for low packet-loss. Then the network's users are able to pick and choose (e.g., via an authenticated Web page) which slices control their traffic. For example, users not trusting the research slices might elect to have all of their traffic controlled by Alice's production slice. By contrast, an early adopter, Doug, might be more interested in the new slices and elect to have his gaming traffic be controlled by Cathy's slice, his HTTP traffic be controlled by Bob's slice, and the rest of his traffic (e.g., VoIP) controlled by Alice's production slice. The key point is that the FlowVisor would enforce isolation between these slices so that if one slice had a problem (e.g., created a forwarding loop), it would not affect the other slices even though they shared the same physical hardware. Further, new services would be deployed more incrementally, i.e., user by user, creating a more graceful service introduction process.

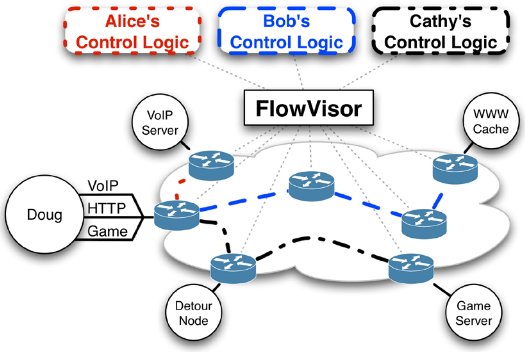

Figure 1: Network slicing allows multiple network protocols to run safely together on the same physical infrastructure.

Slicing Control and Data Planes

Network slicing is a way of allowing multiple services to share control of the same physical network resources. At a high level, the internals of modern switches, routers, base stations, etc. are typically divided into a control plane and a data plane (also called the slow path and the fast path, respectively). The control plane is a collection of software applications typically running on a general-purpose CPU, where the data plane is one or more application-specific integrated circuits (ASICs). The control plane is responsible for formulating higher-level forwarding rules of the form "if a packet matches *pattern,* then apply *actions,*" which are then pushed down to, and enforced by, the data plane. The exact nature of the pattern and actions varies by device: for example, on a router, the pattern might be a CIDR-prefix and actions would be "decrement TTL and then forward out port 14." The important point is that there is a communications channel between the control and data planes.

Similar to how a hypervisor sits between multiple virtual machine operating systems and the underlying hardware, network slicing is a layer between the multiple control planes and the underlying data planes. Each slice runs its own logical control plane and makes its own packet-forwarding rules. The slicing layer ensures isolation between slices by verifying that forwarding rules from different control planes/services do not conflict. Note that once a rule is pushed into the data plane, packets are forwarded at full line speed, so network slicing has no packet forwarding performance penalty. Network slicing can even isolate bandwidth between slices by mapping a slice's actions onto a per-interface quality of service (QoS) queue.

Our network slicing implementation, FlowVisor, is implemented on top of OpenFlow. OpenFlow is an open standard for controlling data planes of existing network hardware. An already existing and deployed switch or router can be OpenFlow-enabled with a firmware upgrade. Once a network device supports OpenFlow, network administrators or researchers can write their own control logic to make low-level packets forwarding decisions, e.g., writing a new routing algorithm. In OpenFlow, the control plane is moved off the network device to an external *controller* (typically, a commodity PC); the controller talks to the data plane (over the network itself) using the OpenFlow protocol. The controller is simply a user-space process that speaks the OpenFlow protocol to OpenFlow-enabled devices.

The FlowVisor acts as a transparent OpenFlow proxy, sitting between the switch and a set of OpenFlow controllers. The FlowVisor intercepts messages as they pass between switch and controller, and rewrites or drops them to ensure that no service violates its slice configuration. FlowVisor's configuration file specifies which sets of resources are controlled by each slice, including topology and bandwidth, and which classes of packets each slice manages.

Deployment and Scalability

Even though it is still a research prototype, FlowVisor has been deployed in various capacities on eight campuses and on one national backbone provider's network. At Stanford University, for example, FlowVisor runs on two different VLANs of the physical production network, including 15 wired switches and 30 wireless access points. It has been in place for over one year and slices the network that the authors use for their daily email, Web traffic, etc. On each of the seven other campuses (including Georgia Tech, University of Washington, Clemson University, Princeton, Rutgers, the University of Wisconsin, and the University of Indiana), FlowVisor manages a testbed network, but there are plans underway to move to the production network. Additionally, the National Lambda Rail (NLR)—a national backbone provider—has deployed OpenFlow and FlowVisor on a dedicated five-node nationwide circuit.

Recently, FlowVisor-sliced networks were showcased at the ninth GENI Engineering Conference. Five distinct OpenFlow-based projects ran simultaneously on the same physical network nodes, as contributed by the eight campuses and NLR. This demonstration is evidence that FlowVisor-style network slicing has the necessary isolation capabilities to test new research on commercially available production equipment.

We also evaluated the FlowVisor in terms of its scalability and overhead. The FlowVisor's total workload is the product of the number of switches, times the average number of messages per switch, times the number of slices, times the average number of rules per slice. Our Stanford deployment does not produce a measurable load on our deployed FlowVisor, so we instead created a synthetic workload that is comparable to the *peak* rate of a published real-world 8000-node enterprise network [5]. Using this synthetically high workload, the FlowVisor maintains under 50% CPU utilization on a single process on a modern server. In terms of performance, we find that FlowVisor adds an average of 16 milliseconds of latency for setting up a new flow and no overhead for additional packets in a flow. Thus, we believe that a single FlowVisor instance could manage a large enterprise network with minimal overhead.

Conclusion

FlowVisor-style slicing combined with OpenFlow offers potential relief to operators and researchers looking to deploy new network services without sacrificing network stability. Our current efforts are focused on expanding our deployments and better "bullet-proofing" isolation between slices. The source code for FlowVisor is freely available from http://www.openflow.org/wk/index.php/FlowVisor.

References

[1] D. Erickson et al., "A Demonstration of Virtual Machine Mobility in an Openflow Network," in *Proceedings of ACM SIGCOMM (Demo),* August 2008, p. 513.

[2] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving Energy in Data Center Networks," *7th USENIX Symposium on Networked Systems Design and Implementation (NSDI '10).*

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, April 2008, pp. 69–74.

[4] http://www.noxrepo.org.

[5] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney, "A First Look at Modern Enterprise Traffic," in *Proceedings of the Internet Measurement Conference 2005,* pp. 15–28.

[6] http://www.quagga.net.

[7] R. Sherwood, G. Gibb, K.-K. Yap, M. Cassado, G. Appenzeller, N. McKeown, and G. Parulkar, "Can the Production Network Be the Test-Bed?" in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI '10),* pp. 1–14.