OpenFlow On Demand

Jeff Chase
Department of Computer Science
Duke University
{chase}@cs.duke.edu

September 5, 2011

Abstract

This working paper outlines plans for combining OpenFlow and cloud infrastructure services. The purpose is to enable dynamic on-demand launch of OpenFlow services in a federated multi-layer cloud network for the purpose of managing traffic passing through the cloud network. We call this "OpenFlow on Demand". As an example, GENI requires OpenFlow in a Slice: it must be possible to launch an OpenFlow "experiment" as a named service in a dyamic GENI slice, and allow other GENI slices to opt-in to that named service to manage their traffic. This note defines the structure of OpenFlow on Demand, outlines motivating scenarios, frames issues relating to authorization and stitching, and outlines the implementation of OpenFlow on Demand in the ORCA control framework.

This is a draft.

1 Introduction

This note address the question of how GENI control frameworks should support dynamic instantiation of OpenFlow services. An *OpenFlow service* is any service that includes one or more OpenFlow controllers to manage network traffic.

More generally, we address the question of how to run OpenFlow services on cloud infrastructure and enable them to manage the traffic of other cloud applications. Cloud infrastructure services and OpenFlow are two key elements of the GENI project. How should we bring them together? This goal presents an interesting challenge for the authorization and stitching architecture of ORCA and other GENI control frameworks, particularly when combined with federation and dynamic circuits.

Vision statement and goals. We plan to demonstrate dynamic on-demand launch of OpenFlow services in a federated multi-layer cloud network for the purpose of managing traffic passing through the cloud network. The primary goal is to launch a distributed OpenFlow service in a GENI/ORCA slice across multiple aggregates, and enable other GENI/ORCA slices to opt-in to have their traffic managed by the OpenFlow service. A secondary goal is to support dynamic deployment of OpenFlow infrastructure services, which we call OpenFlow network managers. An *OpenFlow network manager* is a *trusted* OpenFlow service that is empowered by a local management domain (aggregate) to manage network traffic without further authorization checks. An example of an OpenFlow network manager is a FlowVisor, which runs with full authority to control a switch.

Opt-in. In general, we presume that the customers of an OpenFlow service are the sources and/or destinations of the traffic that the service manages, and that these customers are other GENI/ORCA slices that opt in to (subscribe to) the service. It is also possible to set up OpenFlow services that advertise to individual users and request direct user opt-in. Indeed, direct user opt-in may be a requirement for any experiment

involving mobile computing with real users. But direct user opt-in presents additional challenges for the authorization mechanism, which we do not consider here.

Note that an OpenFlow network manager is a special case that does not require any opt-in. The customer of an OpenFlow network manager is the network owner, who authorizes the service to manage traffic on its network transparently to the sources and sinks, i.e., without the knowledge or consent of the network's users. This is one way to support OpenFlow experiments involving mobile computing without direct user opt-in. Network managers are easy to support: the key issue is the authorization policy.

2 Motivating Scenarios

We consider several motivating scenarios that build on one another. Our goal is to support the key capabilities needed to demonstrate all of them.

OpenFlow federation. Consider a network of autonomous OpenFlow-enabled cloud sites (aggregates) that are members of a federation offering federation-wide cloud services with some common management. An example is a GENI Racks testbed spanning multiple campuses, with OpenFlow switches at each site, running an federated ORCA deployment with a common broker. To support launch of multiple OpenFlow services on the federated cloud, each site must install and maintain FlowVisors for its OpenFlow switches. Managing FlowVisors may be a burden for the aggregates, and may limit testbed-wide monitoring and control (e.g., kill switch).

To simplify management in an OpenFlow federation, we propose a federation-wide network manager. Some central entity deploys a trusted OpenFlow-based network manager service in a slice spanning all participating sites. The network manager service instantiates FlowVisors as needed for the switches at each site, together with related monitoring and control software. The network manager is a layered service above the basic testbed stack, deployed dynamically as a privileged slice. Site authorization policies must trust the manager to control the local OpenFlow switches, e.g., based on delegated trust from a federation root.

Campus isolation. An OpenFlow network manager may enforce policies necessary for the safe functioning of a cloud site or federated testbed. For example, it may isolate untrusted slices from campus networks interconnected to the site. Suppose that a campus offers IP transit service to a testbed site that hosts local and external users. Access to and from the campus network is convenient for local users to connect directly into their slices, and also for hosted experiments that are specially authorized to access the campus network. But other slices may not be permitted to access the campus network. An OpenFlow network manager can block any traffic from entering the campus network from the testbed, if the traffic does not conform to a trust policy. An OpenFlow network manager could provide a more general isolation service along the lines of the once-proposed GENI Gateway Facility.

Data plane as a service. Suppose that the federated testbed sites are interconnected by dynamic circuit networks such as BEN, NLR, ESNet, and/or Internet2. Hosted slices may request dynamic circuits for a private data plane. In some cases it may be sufficient for slices to share a common data plane. Sharing the data plane uses resources more efficiently and offloads management of the data plane from the slice owners. An OpenFlow service can manage and mediate access to a common data plane, as a service to other slices that opt-in as its customers. To do this, the service instantiates a data plane and intercepts customer traffic, redirecting it through the data plane. A federated testbed should support multiple dataplane services, including experimental services, running side-by-side in the testbed. OpenFlow makes this possible.

Programmable packet routing service. As an example dataplane service, consider a testbed-wide packet routing service. Suppose that each site (aggregate) assigns network addresses to virtual nodes at the site, and that these addresses are allocated from a common address space that is partitioned across the sites. We envision using RFC 1918 addresses for this purpose. In principle it could be any address space, but we use IP addresses for interoperability with ARP, DHCP, and commercial OpenFlow switches. An OpenFlow service deploys a point-of-presence (PoP) at each site, links its PoPs with a built-to-order data plane of dynamic circuits, and offers a routing service to its customers. The service uses OpenFlow to intercept customer

packets at their source PoP, and route them through the service data plane to their destination PoP. The receiving PoP injects the packet into the network at the local site, on a suitable VLAN if required. The local network delivers the packet to its destination using standard mechanisms (ARP and Ethernet spanning tree).

3 Understanding OpenFlow

This section summarizes OpenFlow from the viewpoint of an individual network switch within some management domain, e.g., a GENI/ORCA aggregate with an Aggregate Manager (AM). For clarity it is necessary to define some OpenFlow terms more precisely, and introduce some new terms.

Traffic (packets) enter a network switch. Traffic is data. Traffic belongs to somebody, i.e., some other user.

Packets enter and exit a switch on ports, wires, fibers, and channels. These are infrastructure/substrate resources: they carry and manipulate data (packets). Allocation of these resources is governed by the AM.

OpenFlow makes it possible for one or more external controllers to *interpose* on subsets of packet traffic passing through the switch. To interpose on a packet means to control what the switch does with the packet. The controller may also influence how resources of the switch are used to handle the packets.

3.1 Flows and Flowspace

Each packet passing through a switch has a header with fields. These include layer-2 headers (source and dest MAC, VLAN tag, MPLS label), layer-3 headers (source and dest IP), and layer-4 headers (transport protocol ID and port). Each header field can take values drawn from some domain of values, e.g., the set of possible VLAN tags. A *flow* is some possible combination of values of any subset of header fields. Flows correspond to groups of packets that have header fields matching these values. In other words, flows are groups of packets that are related in some way, e.g., because they all come from the same source and/or have the same destination, and/or they are within the same LAN or subnet, and/or they use the same protocol. Flows defined in this way may overlap: any given packet may belong to multiple flows.

The space of all possible combinations of values of header fields is called *flowspace*. Flowspace is the Cartesian product of the domains of all header fields. Each packet is associated with a point in flowspace. Each flow corresponds to a region of flowspace. A stream of packets in the same flow are all within the same region of flowspace—the region corresponding to the flow.

3.2 Controllers

An OpenFlow controller is a running program, e.g., within in a virtual machine. An OpenFlow controller uses the OpenFlow protocol to interact with an OpenFlow switch, possibly through a *proxy* called FlowVisor. Each switch has at most one directly attached controller, which may be a FlowVisor. If the switch's controller is a FlowVisor, then other controllers may also act as a controller for the switch, using the switch's FlowVisor as a proxy. Controllers that use a proxy are called *guest controllers*.

A controller may be a controller for multiple switches. This set of switches is the controller's *domain*.

An OpenFlow controller may interpose on regions of flowspace. That means that the controller can control how the switch handles traffic associated with those regions. The OpenFlow documents refer to the set of regions of flowspace that a controller may interpose on as its *slice* of flowspace. This use of "slice" is related to the GENI/ORCA concept of a slice, as discussed below.

A controller interposes on traffic in its flowspace by installing flow rules in the switch. The flow rules specify patterns to match against the header values of packets passing through the switch, and actions to take on

all packets matching the patterns. The set of allowable actions enable the controller to observe and/or drop or redirect packets in its flowspace. Redirection is accomplished by installing flow rules that transform fields in the packet headers.

3.3 Proxy: FlowVisor

If a controller is a guest controller, then its proxy may restrict the controller's flowspace. If the guest controller requests its proxy to install a flow rule, the proxy checks the rule against the guest controller's authorized flowspace. The proxy does not allow a guest controller to affect any traffic outside of its flowspace.

The proxy may also restrict a guest controller's view of the substrate topology. The OpenFlow protocol defines an interface for a controller to query a switch for topology information. Let us define a controller's *sandbox* as the subset of ports (or other substrate resources) that the proxy reveals to it. A controller cannot redirect packets outside of its sandbox.

The proxy can also hide packets from the controller by restricting the subset of traffic that is checked against the controller's installed flow rules. Specifically, the proxy can choose to hide any traffic arriving through some set of input ports from the controller. It may be able to hide traffic from the controller in other ways. Let us define the controller's *filter* as the subset of its flowspace that the proxy reveals to it. We might also call this filtered traffic or filtered flowspace. Because the proxy can hide traffic based on its arrival port or other topology, it is tempting to think of this filter as an extension of the sandbox. But the sandbox is a restriction on the resources that the controller can affect, while the filter is a restriction on the data (packets) that a controller can affect.

3.4 Flowspace and Slices: Allocation or Authorization?

The OpenFlow documents speak of flowspace as a resource, and of a controller's flowspace as its "slice". They are using the term "slice" somewhat differently than standard GENI/ORCA usage. It is important to draw a distinction between two ways that an entity can "have" flowspace.

A GENI/ORCA slice corresponds to a set of resources. Resources are allocated exclusively to a slice by resource providers, e.g., Aggregate Managers (AMs). An OpenFlow slice corresponds to a set of *data*: other people's packets. Adding resources to a GENI/ORCA slice is a resource allocation problem. Adding traffic to a controller's flowspace is an authorization problem. It is also a stitching problem.

However, any GENI/ORCA slice does have flowspace allocated to it as a resource. This flowspace comprises the set of header values for traffic on the slice's data plane, including any VLANs and/or IP address space assigned to the slice. To avoid confusion, we refer to this flowspace as the slice's *allocated flowspace*. In general, packets in a slice's allocated flowspace are either generated by the slice or destined for the slice, or both. A slice's *source flowspace* is the subset of its allocated flowspace that identifies traffic originating within the slice. A slice's *destination flowspace* is the subset of its allocated flowspace that identifies traffic destined for the slice.

An OpenFlow service is authorized to interpose on some or all of the allocated flowspace of its customers. This means that controllers in the service can install flow rules with patterns that match packet traffic to each customer (matching the customer's destination flowspace) and/or from each customer (matching the customer's source flowspace) as required for the OpenFlow service to manage that traffic. We refer to this flowspace as the service's *authorized flowspace*. If the service is running within a GENI/ORCA slice, we can say that it is the slice's authorized flowspace.

Summary. Every GENI/ORCA slice has allocated flowspace for its own data plane. An OpenFlow-on-demand service may run within a GENI/ORCA slice. In addition to its allocated flowspace, an OpenFlow service (and its slice) has authorized flowspace that defines the traffic it can interpose on. When OpenFlow documents refer to a controller's "flowspace" or its "slice", they are referring specifically to its authorized

flowspace. In speaking of a slice's "flowspace" we should distinguish between its allocated flowspace and its authorized flowspace.

4 OpenFlow on Demand: OpenFlow in a Slice

An OpenFlow-on-demand service is an OpenFlow service, including one or more controllers, running within a GENI/ORCA slice. The slice is allocated for the service via cloud infrastructure services, such as GENI/ORCA aggregates (AMs). Other slices may opt-in to use the service to manage their traffic.

An OpenFlow service and its slice may span multiple cloud sites or aggregates. In this section we consider a service from the point of view of an individual aggregate (AM). The AM controls some substrate in a domain, possibly including one or more OpenFlow switches.

Concepts of operation. A service owner instantiates an OpenFlow-on-demand service by using control tools (e.g., SM) to request resources for a slice and to deploy service software into the slice. The service software includes one or more OpenFlow controllers. The controllers are deployed on slivers (e.g., virtual machines) allocated to the slice within the local aggregate. The control tools may link those nodes to other elements in the slice such as dynamic circuits and/or VLANs.

Once deployed, an OpenFlow service advertises its existence to potential customers in some unspecified way. (E-mail will work for now.) OpenFlow services register within some network-wide name space, e.g., GUIDs, keys, SCIDs, and/or URNs.

Once a service is registered, another application (or experiment or slice) running within the cloud infrastructure may request the cloud service to attach it as an opt-in customer of the OpenFlow service. The customer specifies the name and/or public key of the OpenFlow service it subscribes to.

An opt-in subscription request declares whether the service is to manage the customer's input traffic, its output traffic, or both. Each customer slice can subscribe to at most one OpenFlow service to manage its input traffic, and at most one OpenFlow service to manage its output traffic.

We presume that the opt-in subscription request is made to the AM, e.g., as a configuration command or attribute associated with a new slice or sliver. We assume for now that each controller "acts locally": OpenFlow services interpose on traffic only at the edge. Specifically, if a customer subscribes to an OpenFlow service to manage its output, then the service intercepts each packet at a switch that resides in the same aggregate as the source. If a customer subscribes to an OpenFlow service to manage its input, then the service intercepts each packet at a switch that resides in the same aggregate as the destination. This "edge assumption" is important because it implies that each edge attribute can execute the opt-in request using local information about the customer to drive local configuration actions on the subscribed service's local controller(s). Later we consider how to relax this assumption to consider the inter-domain slice stitching case, in which we must configure OpenFlow controllers attached to switches in the "middle" of the network.

Wish list. There may be other cases to consider in the future. For example, a customer slice might declare that it accepts traffic redirected to it by some OpenFlow service, even when that traffic was not originally destined for that customer. We do not consider how to support that case, but we consider its effect on other policies.

It should be possible for a customer to switch over from one OpenFlow service to another, and perhaps even specify a fail-over secondary. If an OpenFlow service fails or terminates, then its customers may perceive this failure as a network outage.

4.1 Launching an OpenFlow-on-Demand Service

For the most part, launching an OpenFlow service in a slice is not different from launching any other slice. In addition to its controller, the service may request other resources from the aggregate such as virtual machines

or circuits. These resources held by the OpenFlow service (and its slice) may be stitched to external resources allocated to the slice at other aggregates. In addition, the OpenFlow service (or more precisely its slice) holds allocated flowspace for its own slice data plane.

Among other services an aggregate may allow instantiation of virtual machines. The service runs its controllers within VMs, and may instantiate other VMs (e.g., virtual routers) linked to its data plane. These steps are initiated by one or more requests to the local AM. In handling these requests, the AM allocates resources including flowspace for the slice's data plane, initiates loading of any required images or programs to run in the slice, initializes and connects the resources, and returns one or more manifests (e.g., leases).

But an OpenFlow service is special in that it registers OpenFlow controllers. Someone must authorize these controllers. If a controller is a guest controller, then it must register with a proxy, and the proxy must determine a filter, a sandbox, and authorized flowspace for the guest controller.

Let's focus in on what is happening here. This is a question of integrating FlowVisor with a cloud infrastructure service, in this case an Aggregate Manager. The service's controller uses the OpenFlow protocol to make requests to the proxy (FlowVisor). The proxy acts as the policy enforcement point for the controller's filter, sandbox, and authorized flowspace. When the controller requests to register a flow rule at the switch, the proxy checks the request against the filter, sandbox, and authorized flowspace. When the controller interrogates the switch topology, the proxy hides from the controller any topology the switch reports that is not part of the controller's sandbox. In fact, the purpose of FlowVisor is exactly to provide these mechanisms. But where do the policies come from?

4.2 AM Control of OpenFlow Proxies

These new requirements for an OpenFlow-on-demand slice must be governed by new authorization and stitching mechanisms and policies within the AM, which is the ultimate policy decision point for these policy enforcement mechanisms in the proxy. Let's take the new requirements one at a time:

- **Registering a controller.** In general, registering an OpenFlow controller is a harmless action, as long as the proxy does not permit the controller to register flow rules without proper authorization for the flowspace that those rules cover. Even so, an aggregate might impose some authorization policy on requests to instantiate OpenFlow controllers on switches within the aggregate. We can implement this as an ABAC policy at the AM. The AM might delegate authorization policy decisions to the proxy, but we do not consider that case.

- **Filter and sandbox.** We focus on a simple case which we believe to be the common case: OpenFlow controllers in an aggregate can manage any traffic passing through the aggregate's substrate, if the traffic passes through a switch in the controller's domain. The rationale is that the sources and sinks of the managed traffic—e.g., VMs belonging to customer slices—could be instantiated anywhere within the aggregate, so any OpenFlow service that operates in the aggregate operates throughout the aggregate. In this case, the filter and sandbox are used only to isolate specific links or nodes that are not within the aggregate, but are connected to switches managed by the aggregate. The AM substrate description describes the topology within the aggregate's domain. From this description (NDL) we can derive a filter and sandbox to apply to all controllers operating within the aggregate. They change only when the substrate description changes.

  There may be other cases in which an aggregate might use different filters and/or sandboxes for different OpenFlow controllers. For the moment we do not consider these cases. One case to consider in the near future is when the OpenFlow service distributes traffic to local logical QoS queues for different classes of service. The local queues available to it are defined as part of its sandbox.

- **Authorized flowspace.** A controller gains authority to interpose on flowspace only when a customer opts in to the controller's OpenFlow service. Specifically, the controller receives authority to interpose on some subset of each new customer's traffic. We can say that the customer's allocated flowspace is

delegated to (added to) the controller's authorized flowspace. The AM must handle this delegation, since it is authoritative for information about each slice's allocated flowspace and the details of a customer opt-in request.

- **Redirection.** To implement redirection, a controller requests the proxy to register a flow rule that transforms matching packets in a way that changes their destination. The proxy must decide whether to approve the request. It can approve the request if the transformed packets are within the controller's allocated flowspace, i.e., the allocated flowspace of the containing service's GENI/ORCA slice. In other words, an OpenFlow service is permitted to redirect packets matching its authorized flowspace onto its own slice dataplane. The proxy can also approve the request if the transformed packets are within the destination flowspace of a customer that accepts input redirected from the OpenFlow service. Under no circumstances may the proxy allow a controller to register a flow rule that redirects packets outside of the controller's sandbox.

In all cases it the AM that governs the restrictions enforced in the FlowVisor proxy. It is therefore necessary to create a secure *AM-to-FlowVisor Control channel*, and devise a protocol for this channel.

4.3 Stitching a Slice to an OpenFlow Service

The AM must use the AM-to-proxy control channel to inform each proxy within its domain of certain decisions made in the AM. Specifically, it commands the proxy to change its configuration for a guest controller for the following events:

- **Register controller.** If a new controller is approved, the AM passes the proxy a descriptor for the controller's allocated flowspace: it is the AM's view of the allocated flowspace of the OpenFlow service slice that contains the controller, which corresponds to the slice's data plane within this aggregate. The proxy permits the controller to redirect traffic in its flowspace onto the service's slice data plane.
- **Opt-in to manage input traffic.** If an OpenFlow service is to manage the input traffic of a customer slice, then the AM passes the proxy a descriptor for the customer's destination flowspace to add to the controller's authorized flowspace.
- **Opt-in to manage output traffic.** If an OpenFlow service is to manage the output traffic of a customer slice, then the AM passes the proxy a descriptor for the customer's source flowspace to add to the controller's authorized flowspace.
- **Customer termination.** If a customer slice of an OpenFlow service is destroyed or terminated, then the AM commands the proxy to remove the customer's allocated flowspace from the controller's authorized flowspace. More precisely, this must occur if the customer slice relinquishes any part of its allocated flowspace for any reason.
- **Change to service data plane.** If an AM approves a change to the data plane of an OpenFlow service, then it informs the proxy of the change. Any newly allocated flowspace is added to the controller's record so that the controller is permitted to register flow rules that redirect packets matching its authorized flowspace into the newly allocated flowspace. It is harder if the service relinquishes any of its allocated flowspace, since this may require shooting down flow rules that are already registered.
- **Service/controller termination.** If a controller terminates while it still has customers, then it may be necessary to drive configuration actions to fail-over to another service or controller.

4.4 ORCA OpenFlow Handler

In ORCA, these operations are implemented in an OpenFlow handler. An OpenFlow handler exists for ORCA, written by Namgon Kim. The secure AM-to-proxy control channel is based on the AM's ability to

write configuration files into a Unix file system shared with a FlowVisor proxy controller. The AM running the OpenFlow handler rewrites configuration files of the proxy, and restarts the proxy (HUP). This handler was used to demonstrate an ORCA-hosted OpenFlow service at GEC-5 in 2009.

For inter-slice stitching, the OpenFlow handler must be invoked as one step in initializing the customer slice. An ORCA handler is invoked once for each sliver setup. But it may only be necessary for the AM to issue the proxy commands once for each customer slice. Also, the OpenFlow handler should not interfere with the sliver handler (e.g., EC2 handler) that sets up each customer VM. To do this cleanly, we might add a concept of a handler that operates at slice scope, and/or stackable handlers.

The OpenFlow handler must read the allocated flowspace of the slice, and create a descriptor that is understandable to FlowVisor. The descriptor must reflecting the details of the opt-in request, as outlined above.

5 Inter-Domain Stitching for OpenFlow

It remains to consider the inter-domain slice stitching case. In this scenario, a controller registers for an OpenFlow switch that is in a different domain from the sources and/or sinks of traffic passing through its authorized flowspace. By design, the switch's proxy is in the same aggregate as the switch, and is governed by an AM that also controls the switch. But the intercepted traffic enters the aggregate through transit links from other aggregates.

Concrete scenario. A customer subscribes to an OpenFlow service. The OpenFlow service registers a controller for an OpenFlow switch. The switch resides within an aggregate, and the switch and its proxy are governed by the local aggregate's AM. Call this the "OpenFlow aggregate". Suppose further that the customer's edge nodes reside in different aggregates, and the customer slice holds no resources within the OpenFlow aggregate. In this case, the OpenFlow aggregate's AM has no knowledge of the customer slice or of the customer's opt-in request.

The important thing to recognize about this scenario is that it is a minor variant of a standard case for the ORCA stitching architecture. The customer slice's allocated flowspace descriptors are labels (e.g., subnet masks). These labels are produced by other aggregates: the aggregates hosting the customer's edge nodes. The OpenFlow aggregate is a consumer of those labels. The customer's Slice Manager (SM) requests the OpenFlow aggregate to make the stitch, passing manifests (leases) issued by the edge aggregates. The OpenFlow aggregate can validate the stitching labels because the manifests are signed. The dependency graph is similar to standard circuit-stiching cases: the labels must be produced before they can be consumed.

What is tricky about inter-domain slice stitching is that we need some way to specify the opt-in request to the SM, e.g., using the GENIAPI and/or by encoding it in NDL requests. The information must be presented in a way that allows the SM to identify the OpenFlow aggregates that are the stitching points for the selected OpenFlow service. The SM must also inform the AM of each OpenFlow aggregate of the details of the opt-in request (input or output). The OpenFlow aggregates must have a way to validate the public keys of the edge aggregates.

6 Trusted Network Managers

We must also consider the authorization architecture for trusted network managers. In this scenario, an AM is requested to authorize flowspace for a controller that will empower the controller to intercept traffic without the knowledge or consent of the traffic owners.

For example, an externally supplied FlowVisor is an example of a trusted network manager. (See the "OpenFlow Federation" motivating scenario.) The FlowVisor is expected to act as a proxy for other OpenFlow

services, and to limit their scope as discussed in this document. The FlowVisor is empowered to interpose on any traffic passing through the switch.

The mobile user scenario also requires a trusted network manager.

We believe that these cases can be handled in a simple and flexible way using the ABAC authorization architecture. Right?