Meridian: An SDN Platform for Cloud Network Services

Mohammad Banikazemi, David Olshefski, Anees Shaikh, John Tracey, and Guohui Wang,
IBM T. J. Watson Research Center

ABSTRACT

As the number and variety of applications and workloads moving to the cloud grows, networking capabilities have become increasingly important. Over a brief period, networking support offered by both cloud service providers and cloud controller platforms has developed rapidly. In most of these cloud networking service models, however, users must configure a variety of network-layer constructs such as switches, subnets, and ACLs, which can then be used by their cloud applications. In this article, we argue for a service-level network model that provides higher-level connectivity and policy abstractions that are integral parts of cloud applications. Moreover, the emergence of the software-defined networking (SDN) paradigm provides a new opportunity to closely integrate application provisioning in the cloud with the network through programmable interfaces and automation. We describe the architecture and implementation of Meridian, an SDN controller platform that supports a service-level model for application networking in clouds. We discuss some of the key challenges in the design and implementation, including how to efficiently handle dynamic updates to virtual networks, orchestration of network tasks on a large set of devices, and how Meridian can be integrated with multiple cloud controllers.

INTRODUCTION

Today's enterprise information technology (IT) environments must be highly responsive and agile in order to support rapidly changing business requirements. Cloud computing platforms offer a new IT consumption model that enables enterprises to procure computing resources on an as-needed basis and delegate management of the infrastructure to the cloud service provider.

As the number and variety of applications and workloads moving to the cloud grows, cloud service providers have expanded their offerings to include a variety of services beyond basic virtual servers, storage volumes, and network connectivity. Commercial cloud platforms today support a variety of server types (e.g., specialized processing with graphics processing units [GPUs]), multiple storage models (e.g., object, block, or key-value stores), and creation of virtual networks with fine-grained access controls.

Cloud networking features, in particular, have developed rapidly recently. For example, over just a brief period, Amazon's AWS has evolved from providing basic IP connectivity for cloud servers to offering a "virtual private cloud" (VPC), which allows a customer to organize its cloud servers into different subnets with access control rules to govern the traffic that may pass between them. AWS, Microsoft Azure, and other cloud providers also offer virtual private network (VPN) services to securely connect cloud instances back to an on-premises data center.

The OpenStack open source cloud computing platform has also benefited from increased emphasis on networking features, with the Quantum network manager now an integral part of the platform [1]. Quantum initially provided a simple virtual Ethernet switch abstraction with virtual ports that could be associated with virtual interfaces on virtual servers. Each cloud project or tenant could then create its own virtual network instance. More recently, the Quantum networking model in OpenStack has added subnets and IP address management features, with planned extensions for access controls and other more advanced features.

Both AWS and OpenStack (and others) are examples of *network-* or *device-centric* cloud networking models in which users are presented with network-layer constructs such as switches, subnets, and access control lists (ACLs), which they then must configure to create a virtual topology for their cloud application. Moreover, in this model, networking is configured and managed largely independent of other elements of the cloud workload like application images and virtual server groups. This model mimics the traditional siloed nature of IT management in which application and server teams typically have a responsibility distinct from the network administrators, and correspondingly little understanding or control of the network.

While cloud management systems bring all of the virtual IT resources under a single view, the model for managing the network is still primarily designed to provide a virtualized version of similar components and functions as the physical network in the customer's data center. In con-

 0163-6804/13/$25.00 © 2013 IEEE

trast, we describe in this article (and in prior work [2]) a *service-level* model of cloud networking, in which connectivity and associated policies and functions are more fully integrated into the process of provisioning and managing cloud applications. With the emergence of DevOps [3] in the cloud, in which application development and IT infrastructure and operations are much more tightly integrated, we believe networking services and capabilities should be exposed using higher-level abstractions than the device-centric view used in traditional networking.

The emergence of the software-defined networking (SDN) paradigm provides a new opportunity to more seamlessly integrate application provisioning in the cloud with the network through programmable interfaces and automation. With cloud applications demanding greater control over the network, SDNs are a natural fit, whether in infrastructure as a service (IaaS) or platform as a service (PaaS) clouds, private or public. A number of SDN solutions have been proposed for creating virtual networks in multitenant clouds based on standard protocols such as OpenFlow (e.g., in [2]), or using encapsulation and overlay networks (e.g., as described in [4, 5]), and commercial solutions such as [6].

These SDN-based cloud networking solutions have their respective advantages (e.g., in terms of scalability, flexibility, or performance), but they support only certain types of network environments, or specific models of cloud network orchestration. For example, with overlay virtualization, network tunnels and policies are managed at the edge of the data center in software-based virtual switches located in hypervisors. This is a well suited solution for very large-scale environments that only require multitenancy and logical isolation, but does not allow finer-grained control over network paths to achieve goals such as fast failover or traffic prioritization. OpenFlow provides this level of fine-grained control through its programmable interface for packet handling and forwarding in physical and virtual switches, but full exploitation of this flexibility is made challenging by a number of practical issues such as hardware flow table limitations or controller performance [7]. And for legacy environments that use traditional switches, virtual networks for cloud application may be implemented simply as virtual LANs (VLANs).

Similarly, there are a variety of cloud orchestration platforms that each provide their own model and application programming interface (API) for virtual networks. With an SDN-based approach, we have an opportunity to provide a common service model and programming interface for these cloud controllers. This makes it possible to ensure some consistency in the way virtual networking is configured for applications deployed in the cloud.

In the remainder of this article, we describe Meridian, an SDN-based framework that supports a service-level model for application networking and can exploit multiple options for implementing virtual networks on the underlying physical network. We discuss the SDN controller architecture in the context of cloud networking services, and how the architecture lends itself to deployment on different types of underlying network environments. In addition, we describe some key issues arising in Meridian's implementation, in particular orchestration of network tasks, and dynamic updates to the virtual network topology, as well as our experience in implementing Meridian to work with multiple cloud orchestration platforms. Our focus is on the use of Meridian in the context of enterprise applications that require flexible virtual network services to support a variety of application topologies.

The Meridian cloud networking platform architecture design is inspired by the emerging conceptual models of software-defined networking. It is organized as three main logical layers: network model and APIs, network orchestration, and interfaces to underlying network devices.

SDN Architecture for Cloud Networking

The Meridian cloud networking platform architecture design is inspired by the emerging conceptual models of SDN described, for example, in [8, 9]. A high-level view of Meridian's architecture is shown in Fig. 1a. It is organized as three main logical layers: network model and APIs, network orchestration, and interfaces to underlying network devices. Network applications are at the top of the stack, as consumers of the APIs. Below, we briefly describe the function of each of the layers in turn.

Abstract API Layer

The goal of the abstract API layer is to present applications with a network model and associated APIs that expose only the information needed to interact with the network. For example, the single switch model described earlier is a simple model for a provisioning system that just needs to establish connectivity by logically "plugging in" virtual server interfaces. For a more sophisticated cloud orchestrator that executes a complex placement algorithm, a more complete view of the topology may be necessary in order to place VMs on hosts that are well connected or proximal in the network. Similarly, a control application that manages access to network storage may require a topology view that exposes multiple paths with annotations of utilization or latency with APIs to request redundant high-bandwidth paths.

In Meridian we provide network services to the higher-layer cloud orchestration application using both a *declarative* and *query* API. The API allows the cloud controllers to request policy-based connectivity between logical groups of virtual servers. That is, it can "declare" how it wants the virtual network for a multi-virtual machine (VM) application to be constructed, along with policies for controlling access, prioritizing traffic, or traversing middleboxes. The query part of the API supports requests for abstract topology views, or gathering performance metrics and status for specific parts of the network.

Network Orchestration Platform

The network orchestration layer plays several important roles in the SDN architecture. First, it must perform a logical-to-physical translation of commands issued through the abstraction layer above. Applications interact with a logical view of the network using high-level APIs. The

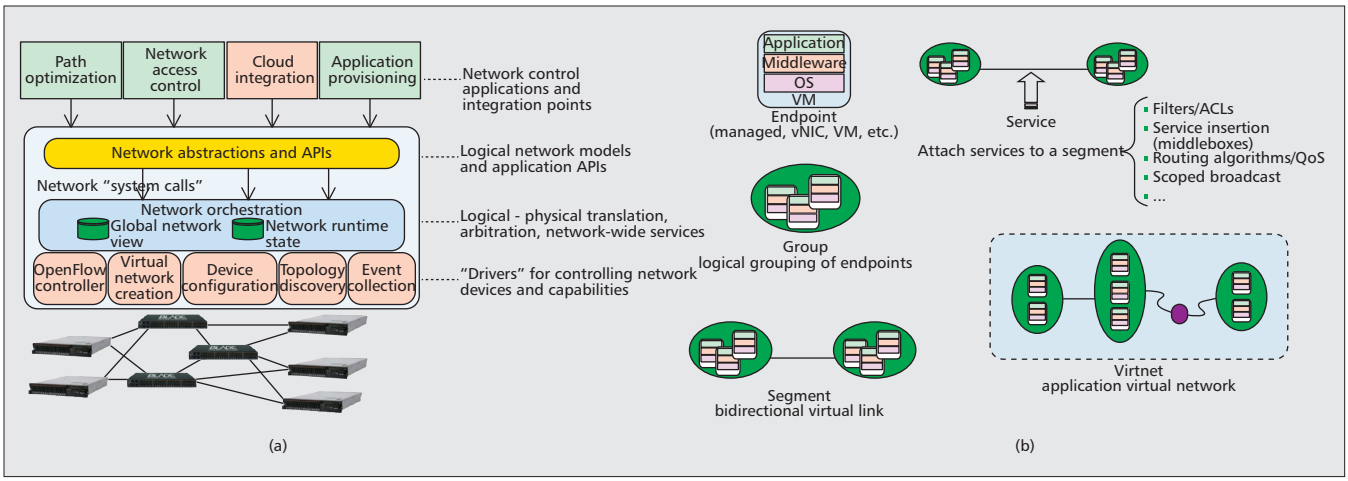

Figure 1. *a) Meridian SDN cloud networking platform architecture; b) Meridian service model.*

orchestration layer must convert these API calls into the appropriate series of commands on the underlying network. For example, connecting a VM interface to a logical switch port may in fact require creation of forwarding and access control rules across several physical switches in the network. Second, the orchestration layer provides a set of network-wide services to applications, such as views of the topology, notifications of changes in link availability or utilization, and path computation according to different routing algorithms. Finally, as its name implies, the orchestration layer provides coordination and arbitration between network requests issued by applications, and mapping of those requests onto the network. This may require selecting between multiple mechanisms available to achieve a given operation, for example, setting up a virtual network using an overlay, or using traditional VLANs.

The Meridian orchestration layer provides a number of key services to support cloud networking, including a global annotated view of the data center topology for applications that require it, different routing algorithms (e.g., standard shortest-path and quality of service [QoS]-based), support for service insertion using virtualized middleboxes, and a planner module that can schedule network configuration or control tasks. These services are described in more detail later.

Applications

The SDN architecture shown in Fig. 1a is intended to support a wide variety of applications that need to control or otherwise interact with the network. We consider two broad categories of applications when designing the interfaces in the abstraction layer and the services in the orchestration layer. *Network control applications* are standalone modules that perform relatively low-level network functions such as path computation and optimization, fine-grained access control, and traffic monitoring and diagnostic operations. *Network integration points* are application modules that provide integration functions that connect higher-level business or IT processes to the network. For example, an enterprise application provisioning process that automatically deploys application servers and storage could use the corresponding network integration point to request automated configuration of appropriate firewall rules and traffic prioritization to support the new application.

With Meridian, our focus is to create an integration point that can effectively connect a variety of cloud orchestration applications to the set of virtual networking services. In our experience, this requires the Meridian cloud integration application to be separated into a common module that provides an identical set of functions and interfaces to any higher-layer cloud platform, and in some cases, a platform-specific component that extends and directs networking operations to Meridian. With OpenStack's Quantum network manager, for example, it is relatively easy to extend it to use Meridian services through its plugin architecture and extensible API layer. In other cases, adding support for Meridian required source-level access to the platform to add or modify the networking-related modules.

Network Driver Layer

The lowest layer of the SDN architecture consists of "plug-ins" or "drivers" that enable the controller to interface with various network technologies or tools. The orchestration layer uses these drivers to issue commands on specific devices, or collect information from the network to build and update its view of the network. In an OpenFlow-capable network, for example, one such driver could provide an interface to an OpenFlow controller that allows insertion of flow rules in physical or virtual switches. Similarly, another module could provide access to data maintained by a traditional network management system that performs topology discovery or provides notification of link or device availability.

In Meridian, we have implemented a logical driver that interfaces to OpenFlow devices to create virtual networks and accompanying services. We are also working on drivers to enable virtual network creation using overlays and to access comprehensive topology data gathered by network management tools such as IBM Tivoli Network Manager (ITNM).

MERIDIAN IMPLEMENTATION

Below we describe some of the details of our implementation of the Meridian cloud networking platform, focusing on the layers of the SDN architecture described earlier. We also discuss how we extend two cloud orchestration platforms with Meridian services.

Our Meridian prototype is built on the open source Floodlight controller platform [10]. Floodlight is a modular Java-based OpenFlow controller that provides OpenFlow protocol support and a number of services such as basic link discovery and routing for OpenFlow-enabled switches. We leverage and extend some of Floodlight's native support, and add new modules to implement the Meridian architecture.

MERIDIAN NETWORK ABSTRACTION MODEL

Figure 1b shows some of the basic constructs of the Meridian service model. This model allows users to think of networking in terms of logical topologies for their cloud-based applications. Rather than exposing device or network-level information, Meridian provides a service-level network model for users to specify logical connectivity and policies or services associated with the virtual links between VMs.

In the Meridian network service model, five types of entities are defined:

- `endpoint`: This is an entity to represent a virtual network interface on a virtual machine.
- `group`: This is a collection of endpoints that share the same connectivity properties; grouping simplifies the application of the same policies on multiple endpoints. Grouping can optionally also imply connectivity between endpoints in the group.
- `service`: an entity to describe the services on a connectivity path. For example, users can define a customized routing policy, filter, or middlebox traversal policies in a service entity.
- `segment`: a network segment is used to specify the connectivity path between two groups, defined by a 3- tuple: $\{g_1, g_2, svc\}$, where g_1 and g_2 are the endpoint groups of the connectivity path, and *svc* is the requested network service(s) on the connectivity path.
- `virtnet`: A virtual network is a logical topology containing groups, segments, and services. Virtual networks can be created for a single cloud tenant, or at finer granularity for a given multi-VM application. A virtual network can be represented by $VN = \{G, S\}$, where G is a set of endpoint groups, and S is a set of segments defined on G. Each segment in S is an edge defined by two endpoint groups $\{g_1, g_2 \in G\}$ and annotated with a specified service *svc*.

Using these entities, users can construct a variety of connectivity topologies among VMs belonging to different applications. Typical enterprise web service applications have a three-tier topology that consists of a web server tier, an application server tier, and a database tier. The communication between different tiers often has different service requirements. Some examples include restricted communication between tiers using firewalls, load balancing between web and application tiers, traffic prioritization and QoS for some traffic classes, and scoped broadcast between specific VMs to enable heartbeat-based failover mechanisms. Figure 2 shows an example for a multi-tier web application with six VMs, grouped into tiers with segments connecting them. The segments have been defined with various services such as middlebox traversal or scoped broadcasting. We can use this model to flexibly construct topologies for many different types of applications.

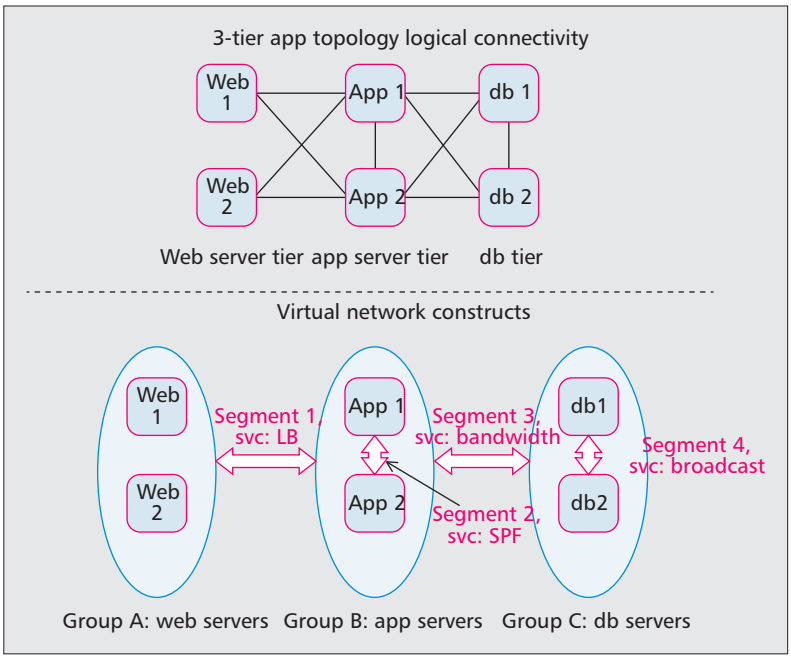

Figure 2. *Constructing a 3-tier application topology using the Meridian network model.*

Meridian REST APIs — Floodlight exposes its services to applications using REST APIs in which applications use standard HTTP requests to send or receive JavaScript Object Notation (JSON) formatted data. Meridian virtual network functionality is implemented in a separate Floodlight module that provides commands for creating, deleting, configuring, and updating individual instances (e.g., a specific group or segment) or collections (e.g., the set of all segments in a virtual network) of network entities using corresponding URLs.

Internally, each instance in the model, no matter what type, is represented as a hash table <key, value>. When using POST to create an instance or PUT to modify an instance, the caller can add fields to the instance simply by including them as <key, value> pairs in the JSON object sent in the request. This allows the caller to augment the instance with additional information such as adding a name, a new endpoint, or a security key. In addition, Meridian supports commands to validate, install, and uninstall virtual network entities. These are useful for higher-layer cloud orchestration systems to first ensure that updates and changes are checked for validity before committing them to the underlying network.

The planner module functions mainly as a scheduler. The plans themselves contain the intelligence to complete the task. Together, plans and the planner provide a mechanism for scheduling work items which can be performed in parallel or sequentially, whichever is required.

Orchestrating the Deployment of Network Services in Meridian

The primary components in Meridian's orchestration layer are the planner and deployer modules that specify and deploy network tasks, respectively.

Meridian Planner Design — Meridian implements network services as one or more plans. For example, when a Meridian client application performs a PUT operation on a virtual network to validate or install it, a plan is executed to perform the task. The planner module functions mainly as a scheduler; plans are posted to the planner to be executed and the planner maintains a pool of threads for scheduling the plans it receives. The plans themselves contain the intelligence to complete the task. Together, plans and the planner provide a mechanism for scheduling work items which can be performed in parallel or sequentially, whichever is required.

Meridian plans are composable in that a plan can start other plans, simply by posting them to the planner for execution. So for example, *Plan1* may be a plan that installs a virtual network, which in turn starts *Plan2*, which installs a specific segment of that virtual network. Since plans can be executed in parallel, *Plan1* may start several plans at once (e.g., *Plan2*, *Plan3*, and *Plan4*) and wait for them to asynchronously complete. This is required for scalability, for example, to configure a large set of switches in parallel rather than one at a time. However, if tasks must be completed in order, *Plan1* may choose to start *Plan2* and wait for the reply, possibly changing its behavior based on the success or failure of each sub-plan in turn.

The composability of plans provides the means to continually grow the set of available plans, similar to building class libraries that can be used and extended by others. Our approach is in contrast to providing a single monolithic orchestrator module, which contains a single optimization program that provides all the services but is difficult to extend.

Combining simple plans into more sophisticated plans led us to categorizing plans into two types:

- *High-level plans* work with Meridian service model instances such as virtual networks, groups, and endpoints.
- *Low-level plans* perform low-level actions and work with existing network objects, such as device objects returned from the Floodlight device manager service.

For example, high-level plans may compute a set of paths, while low-level plans would generate the associated OpenFlow rules for those paths. This provides a level of separation and flexibility with plans using model objects separate from plans, which perform low-level functions. As we enhance Meridian's virtual network model, high-level plans can be extended (or new ones written) that can still use the existing low-level plans. Likewise, low-level plans can be extended or replaced with little or no impact on high-level plans. Time-based scheduling is also supported in Meridian. This allows a plan to schedule itself periodically (e.g., to gather statistics or monitoring data) or a client application to schedule a set of plans for execution at a future point in time.

Deploying Plans in the Network — The planner manages the execution and state of Meridian plans. Specifically, it processes a number of methods that are part of each plan as described below:

- `validate()`: performs a variety of error checks to determine if installation of the plan is likely to succeed and creates the list of network commands (e.g., OpenFlow rules) that are required to actually install the model into the physical network. This method does not make changes to the physical network.
- `install()`: installs the model into the physical network by transferring the list of network commands generated by the `validate()` method to the deployer.
- `undo()`: reverses the install operations performed by the `install()` method of this plan, basically providing a way to uninstall virtual networks.
- `resume()`, `suspend()`: resumes and suspends execution of the plan, respectively.

These methods are hierarchical in nature; that is, when `validate()` is called on a high-level plan, it will call `validate()` on the lower-level plans. It is the same for the other methods.

During execution of these methods, plans go through a sequence of states that are maintained relative to the state of sub-plans. For example, prior to a `validate()` call, the state of a plan is UNVALIDATED. When `validate()` is called, its state is changed to VALIDATING and remains in this state as all sub-plans are validated. Afterward, its state is VALIDATED (or UNVALIDATED if a sub-plan fails to validate).

The basic process flow in the orchestration layer starts with a `validate()` method called on the root plan, which in turn will trigger validation on all plans in the hierarchy. This causes each sub-plan to perform error checking and generate a list of network commands. Problems encountered at lower levels are propagated up so that higher-level plans can determine a course of action during errors or failures. In the second step, `install()` is called on the root plan, which cascades down to all sub-plans. The `install()` method posts the list of configuration commands to the deployer and handles errors. The deployer accepts a stream of network device commands from the `install()` method in each plan, and sends the commands to the appropriate device in the network. Success or failure is returned back to the plan.

The deployer is a multithreaded scheduler, which allows it to send commands to multiple devices in a parallel manner. Being a central point from which network device commands are sent, it has an opportunity to merge or otherwise optimize the set of commands sent to each device.

Separating plan validation from installation does not eliminate the need to handle failures during `install()` processing, but does catch some of the issues that would arise during `install()` processing in advance of making

actual changes to the network devices. It also provides a mechanism for trying sets of commands in a "what-if" manner.

Topology Services

Meridian topology services contribute to the overall goal of realizing cloud virtual networks by providing a view of the dynamic set of underlying network resources. Topology services are used by other essential services, such as routing, and also by higher-level functions built on top of Meridian such as VM placement.

As with all Meridian services, a key goal, and challenge, is flexibility in supporting a large set of use cases, as well as handling a broad array of underlying network devices. The goal of flexibility is well served by representing the network as a simple annotated graph. Network interfaces and links correspond to graph vertices and edges, respectively. Both edges and vertices have a set of associated attributes, with as many as possible designated as optional. For example, edges need not have associated attributes such as bit rate, loss rate, or latency, but these can be added easily if needed. On the other hand, all vertices require a parent node with which the corresponding network interface is associated (nodes could be Ethernet switches, routers, servers, etc.).

The graph model lends itself to a variety of network abstractions, or "views." A summary view may be produced by replacing each vertex/interface with its corresponding parent node. The resulting graph reflects the interconnection of devices without the specifics of individual network interfaces. Additional views of the network may be provided via transformations provided by various graph algorithms.

The topology service goes beyond simple graph representation to provide a framework for integrating potentially overlapping and even conflicting topology information from multiple sources. Meridian's initial topology service implementation gathers information from three main sources, as described below.

First, as mentioned earlier, our topology service retrieves information from Floodlight, including a list of managed switches, a description and list of features for each switch, a list of links between switches, and information related to all known medium access control (MAC) addresses, along with the switch and port to which each address is connected. Although the topology data provided by Floodlight is substantial, it does not include information internal to servers such as hypervisor nodes. For that, libvirt and a set of operating system commands are employed. Libvirt provides a consistent interface for querying and controlling a set of hypervisor implementations including Xen, KVM, VMWare, and others. Meridian's topology service implementation "discovers" hypervisors by monitoring advertisements sent by the libvirtd virtualization daemon. For each hypervisor, the service uses libvirt to query its identity (host name and UUID), and each network interface and VM (including its associated virtual network interfaces). Information on each real network interface is collected using local commands on the hypervisor hosts. No commands or queries are issued to operating systems or applications running inside VMs.

The topology service updates the set of known interfaces and associated nodes based on periodic queries of the Floodlight controller and discovered hypervisors. It computes the set of links between switches and endpoints by correlating the MAC address information provided by Floodlight with that provided by libvirt and host commands. The inferred links include physical links between hypervisor nodes and real Ethernet switches as well as virtual links (e.g., between VMs and virtual switches).

The Meridian topology service is made available via its own set of REST APIs with a comprehensive set of commands that allows information to be provided with the desired level of granularity. For example, the list of all interfaces, only virtual or real interfaces, or a specific interface identifier are supported, along with corresponding interface attributes.

The initial implementation demonstrates the efficacy of the simple node, interface, and link model, and provides a basic framework for integrating information from additional sources. We plan to extend the sources to also include topology information from traditional network management tools, for example, using Simple Network Management Protocol (SNMP).

Meridian Virtual Networks with OpenFlow

Given a defined virtual network topology, Meridian realizes each segment by installing OpenFlow rules on switches. We implement different routing policies to support various services defined on segments. In the current implementation, we are able to support shortest path routing, middlebox waypoint routing, access control filters, and scoped broadcast using corresponding OpenFlow rules. To implement a given segment $\{g_1, g_2, svc\}$, we first locate the attachment points of each endpoint in groups g_1 and g_2 using Floodlight's device manager module and the Meridian topology service. Then, for each pair of endpoints $\{ep_1, ep_2, ep_1 \in g_1, ep_2 \in g_2\}$, we set up the routing between ep_1 and ep_2 using policies defined in the *svc* structure. This way, we have the flexibility to set up routes differently for different endpoint pairs.

A major challenge in Meridian is to manage dynamic updates to the virtual network topology, for example, when adding or removing a segment, adding an endpoint to an existing group, or changing the service for a segment. These updates may be required in response to a user request, or an action taken by the cloud orchestration layer to alter the application topology, perhaps to scale out a particular cluster of servers. To manage these updates, Meridian maintains a control block for each virtual network. Similar to the process control block concept in operating systems, a virtual network control block keeps a record of the network installation operations for the virtual network during its life cycle. A control block is created when each virtual network is defined, and all actions (e.g., OpenFlow rule installations) taken for a virtual network are kept as part of its state

As with all Meridian services, a key goal, and challenge, is flexibility in supporting a large set of use cases, as well as handling a broad array of underlying network devices. The goal of flexibility is well served by representing the network as a simple annotated graph.

We are also exploring additional enhancements to Meridian services, such as the planner to support partial recovery of failed plans and the topology service to add a more generalized topology discovery capability.

in the virtual network control block. Each action is also associated with a context that describes which entities are affected by the action (which group, segment, etc.), and whether the entity is still active (e.g., has not been deleted). When a virtual network is changed, we compute the new actions to update the network installation based on current actions that have been taken. For example, with the list of OpenFlow rules that have been installed for the virtual network entity, we compute a "delta" for the change that allows us to add the new rules, or remove current rules, necessary to effect the change. Using the virtual network control block scheme, we are able to manage most updates of virtual networks incrementally at runtime without tearing down and re-installing the whole network.

Cloud Controller Integration

Below we briefly describe our approach in integrating Meridian with two cloud orchestration platforms. Although these platforms required different approaches for integration, they demonstrate Meridian's ability to offer a uniform set of service APIs to multiple higher-layer cloud provisioning systems.

OpenStack Integration with Quantum — OpenStack is an open source cloud computing platform with three major components: compute, network, and storage [1]. Its networking component, Quantum, has a pluggable, extendable, and API-driven architecture, and is used as part of the cloud operating system to create and manage networks required by cloud applications.

The main Quantum building blocks are networks, subnets, and ports, and the Quantum API is defined to manipulate (create, delete, and update) these entities. We have developed a Quantum plug-in for Meridian, which maps the basic Quantum constructs to the Meridian network model. In particular, when a Quantum network is created, a Meridian virtual network is created, along with an empty group and a segment enabling all-to-all communication within the group. Then the virtual network is installed. As Quantum virtual ports get created on a given network, endpoints are created and added to the corresponding Meridian group.

While this allows the Quantum network manager component to work with Meridian using the standard APIs, it does not exploit the full flexibility of the service-level networking model. For this, we have introduced a new set of extensions to the basic Quantum API and implemented them as a Quantum plugin. As part of future work, we are also investigating extensions to the Nova compute module to support multi-VM application deployments that can better exploit Meridian services.

Integration with A Commercial Cloud Platform — In addition to OpenStack, we also integrated Meridian with the IBM Smart Cloud Provisioning (SCP) cloud controller. SCP uses a fully decentralized architecture with a set of managers, or bots, that handle compute and storage resources for tenants. The managing processes run in a peer-to-peer fashion, with one instance elected to coordinate provisioning operations. In the original SCP system, there are storage bots and hypervisor bots that manage the image and volume, and computing resources, respectively, with a limited model of network services for VMs. To support Meridian network services, we extended SCP with a new network bot type. Network bots are managing processes that handle network configuration requests for tenants. Similar to storage bots and hypervisor bots, network bots operate in peer-to-peer mode with one elected leader. The network leader bot is the primary contact that receives network configuration requests from the SCP web service node and issues requests to the Meridian controller to configure network topology for tenants.

Summary and Future Work

In this article, we describe Meridian, an SDN-based controller framework for cloud networking. We introduce a network service model for users to construct and manage logical topologies for their complex workloads in the cloud. We discuss several key issues in the design of Meridian, including installation and management of virtual networks, how to dynamically update virtual network entities efficiently, and our approach to scalable orchestration of network control and configuration tasks. We also describe how Meridian can be leveraged by the OpenStack and IBM Smart Cloud Provisioning cloud controllers.

Our current Meridian system is only an initial prototype of an SDN-based networking framework for the cloud. There are several challenges remaining in all layers of the architecture that need to be explored in our future work. First, the network service model we describe is mainly focused on managing the connectivity properties of cloud applications like web service applications. However, we would like to explore how it should be extended for performance-sensitive workloads, such as Hadoop or content streaming. These workloads require more specialized routing and traffic delivery support to optimize their performance.

Although Meridian supports multiple instances of virtual networks, more work is needed to fully understand the implications of multitenancy. For example, with many tenants at cloud scale, interactions or conflicts between network configurations must be carefully managed. Multitenancy also requires consideration of multiple virtual networks together in order to jointly optimize the required network device state. Another related challenge is clearly scalability of the platform in order to support a large number of tenant network requests — the amount of state required in network device tables will be a limiting factor without additional schemes to optimize flow rules and other resources.

We are also exploring additional enhancements to Meridian services, such as the planner to support partial recovery of failed plans and the topology service to add a more generalized topology discovery capability (as discussed earlier). Finally, to validate the ability of the Meridian controller to support multiple methods for implementing cloud networks, we are integrating overlay-based virtual networking as an alternate

or complementary mechanism to manage the underlying network.

References

[1] "Openstack Cloud Software," http://www.openstack.org, 2012.
[2] T. Benson *et al.*, "CloudNaaS: A Cloud Networking Platform for Enterprise Applications," *Proc. ACM Symp. Cloud Computing*, Oct. 2011.
[3] M. Loukides, What Is DevOps? O'Reilly Media, 2012.
[4] IETF, "Network Virtualization Overlays," http://datatracker.ietf.org/wg/nvo3/, Sept. 2012, work in progress.
[5] K. Barabash *et al.*, "A Case for Overlays in DCN Virtualization," *Proc. Wksp. Data Center-Converged and Virtual Ethernet Switching*, Sept. 2011.
[6] Nicira, Inc., "Networking in the Era of Virtualization," white paper, 2012, http://www.nicira.com.
[7] A. R. Curtis *et al.*, "DevoFlow: Scaling Flow Management for High-Performance Networks," *Proc. ACM SIGCOMM*, Aug. 2011.
[8] S. Shenker *et al.*, "The Future of Networking, and the Past of Protocols," slides at http://www.slideshare.net/martincasado/sdn-abstractions, June 2011.
[9] Open Networking Foundation, "Software-Defined Networking: The New Norm for Networks," white paper, Apr. 2012.
[10] "Floodlight," http://floodlight.openflowhub.org, 2012.

Biographies

MOHAMMAD BANIKAZEMI [SM] is a research staff member at the IBM T.J. Watson Research Center. His research interests include cloud computing and software-defined networking. He has a Ph.D. in computer science from Ohio State University, where he was an IBM Graduate Fellow. He has received an IBM Research Division Award and several IBM Invention Achievement Awards. He is a senior member of the ACM.

DAVID OLSHEFSKI received his B.S. degree in computer science from State University of New York at Albany, his M.S. degree from Renssalaer Polytechnic Institute at Hartford, and his Ph.D. degree from Columbia University. He has worked at IBM Research since 1988, and his current research focus is on software-defined networking and high-performance networking.

ANEES SHAIKH Anees Shaikh [S '92, M '99] (aashaikh@us.ibm.com) is with the System Networking division at IBM, where he works on software-defined networking. Prior to this, he was a research staff member and manager with the IBM T.J. Watson Research Center in New York, where he led research groups working in data center networking, software-defined networking, cloud computing, and systems management. He received a Ph.D. in computer science and engineering from the University of Michigan, and B.S. and M.S. degrees in electrical engineering from the University of Virginia.

JOHN TRACEY (traceyj@us.ibm.com) is a senior technical staff member at the IBM T. J. Watson Research Center in New York. He received his B.S. and M.S. degrees, both in electrical engineering, from the University of Notre Dame in 1990 and 1992, respectively. In 1996, he joined IBM Research as an advisory software engineer after receiving his Ph.D. in computer science, also from Notre Dame. He has pursued research in the area of system software for networking with a focus on performance and scalability of TCP/IP and web and SIP servers. He has contributed research technologies to multiple IBM products. Currently, he is in the Systems Networking department working on software-defined networking.

GUOHUI WANG (wangg@us.ibm.com) is a research staff member at the IBM T. J. Watson Research Center. Before joining IBM, he received his Ph.D. degree in computer science from Rice University, an M.S. in computer science from the Chinese Academy of Sciences, and a B.S. in electrical engineering from the University of Science and Technology of China. His research interests are in data center networking and cloud computing with a focus on new network architecture, network virtualization, and software-defined networking in cloud data centers.

To validate the ability of the Meridian controller to support multiple methods for implementing cloud networks, we are integrating overlay-based virtual networking as an alternate or complementary mechanism to manage the underlying network.