Network Programmability as a Cloud Service

Project Proposal

1. Vision and Goals of the Project

To implement a mechanism with which Infrastructure-as-a-Service (IaaS) providers can have same kinds of control on the hardware irrespective of ownership of those hardware components.

The primary goal of this project is to provide network resources and their control as a service, which can be used by multiple IaaS providers. The service creates fully virtualized network resources that could be managed and controlled independently.

The users will have a unified view of the network irrespective of the hardware administrative domain in which the devices are placed.

Users:
IaaS providers will use this service

2. Scope and features of the Project

- Offer same control of networking resources over different hardware to IaaS providers
  - Introducing an abstraction layer to support services across multiple hardware domains managed by separate entities
  - Facilitating communication with network devices across multiple hardware domains
  - As an extended goal, provide bandwidth and QoS agreement across multiple tenants in MOC.
- Extensibility: Providing networking services and SLAs to higher level providers and tenants while having different hardware providers

3. Solution Concept

Our task includes creating a multi-tenant SDN network using Mininet

Mininet: Mininet is network emulator to create host/servers, switches and routers. It uses Python to create topologies. Hosts in Mininet behave just like real machine; we can even SSH into it. Host can also transmit packets with a mentioned speed and delay. Switch also processes the packet with queuing.

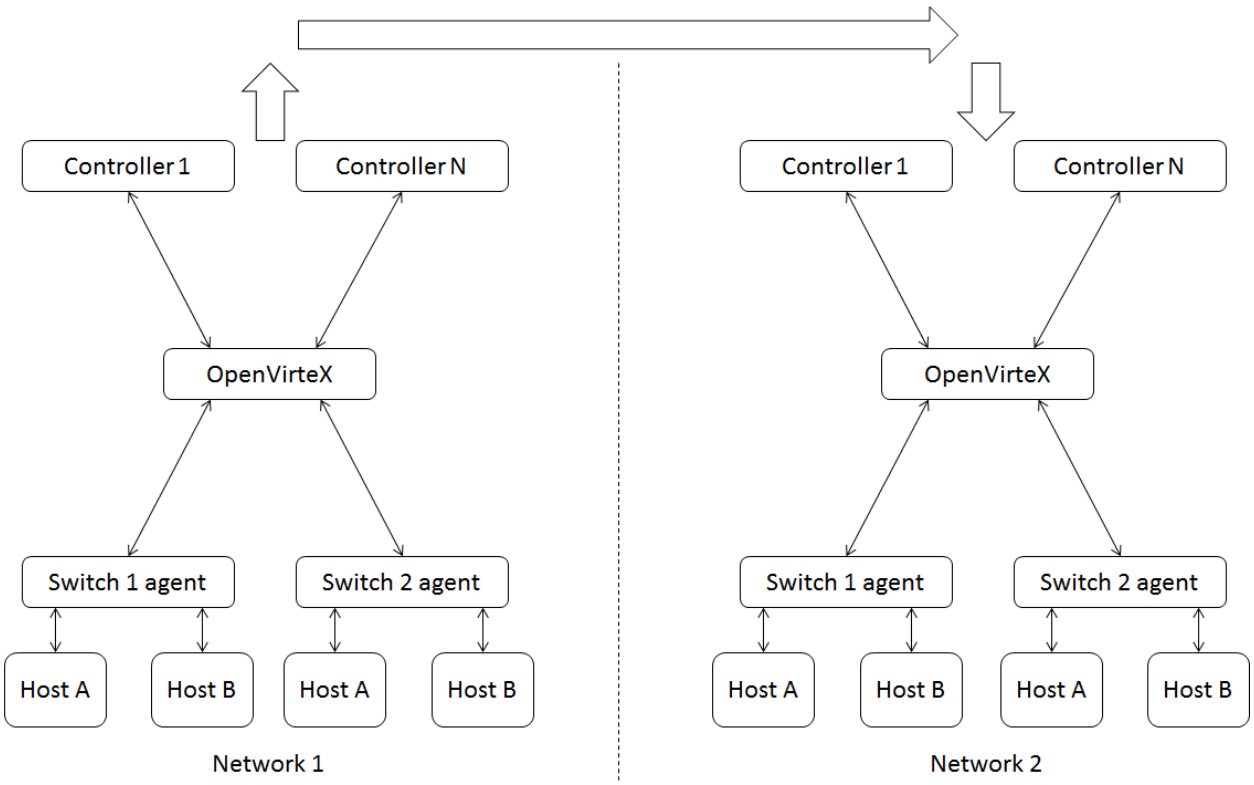

Figure 1: High-Level Network Architecture

Advantages of Mininet:

- We can create custom small to complex large-scale topologies within few seconds. It uses Linux namespace for creation of virtual networks, which is also used for Linux containers in application like Docker.
- We can run any application in Linux machine using that it works as application running from host.
- We can run Mininet on any Linux box or Amazon EC2 cloud.

- Mininet provides flexibility to customize packet forwarding using OpenFlow protocol.
- It is easy to understand and start writing Python scripts and experiment with the different topologies.

Limitations:

- Hosts or switches running on Mininet can't go beyond resources shared by it. For example, on laptop we cannot create links with bandwidth 10 Gbps if it is not supported by physical system.
- Mininet can run on Linux host only. Thus, it can't run applications that support only Windows or BSD OS.
- By default, Mininet remains isolated from LAN. Although, we can use NAT object --nat option to connect to Mininet network via LAN through NAT. Also, we can use real physical switch to connect Mininet interface.

When we start Mininet, it creates instances of controller, switch and hosts by default. The number of hosts connected to each switch can be provided thus customizing the topology.

OpenFlow: OpenFlow is open standard used to control forwarding tables of network devices (switches, routers, access points). It is a standardized protocol for interacting with the forwarding behavior of switches from multiple vendors. It provides us a way to control the behavior of switches throughout our network both dynamically and programmatically. OpenFlow is one of the key protocols in many of the SDN solutions.

How OpenFlow works:

In a classical router or switch, the fast packet forwarding (data path) and the high level routing decisions (control path) occur on the same device. An OpenFlow Switch separates these two functions. The data path portion still resides on the switch, while high-level routing decisions are moved to a separate controller, typically a standard server. The OpenFlow Switch and Controller communicate via the OpenFlow protocol, which defines messages such as packet-received, send-packet-out, modify forwarding-table, and get-stats.

The data path of an OpenFlow Switch presents a clean flow table abstraction; each flow table entry contains a set of packet fields to match, and an action (such as send-out-port, modify-field, or drop).

When an OpenFlow Switch receives a packet it has never seen before, for which it has no matching flow entries, it sends this packet to the controller. The controller then makes a decision on how to handle this packet. It can drop the packet, or it can add a flow entry directing the switch on how to forward similar packets in the future. A remote controller via secure channel controls flow table.

OpenVirteX (OVX): OpenVirteX is a network virtualization platform, which allows you to specify your own topology and addressing while retaining control of your virtual OpenFlow network. In essence, we are introducing the concept of programmable virtual networks. OVX sits in between the physical hardware and the virtual network controllers.

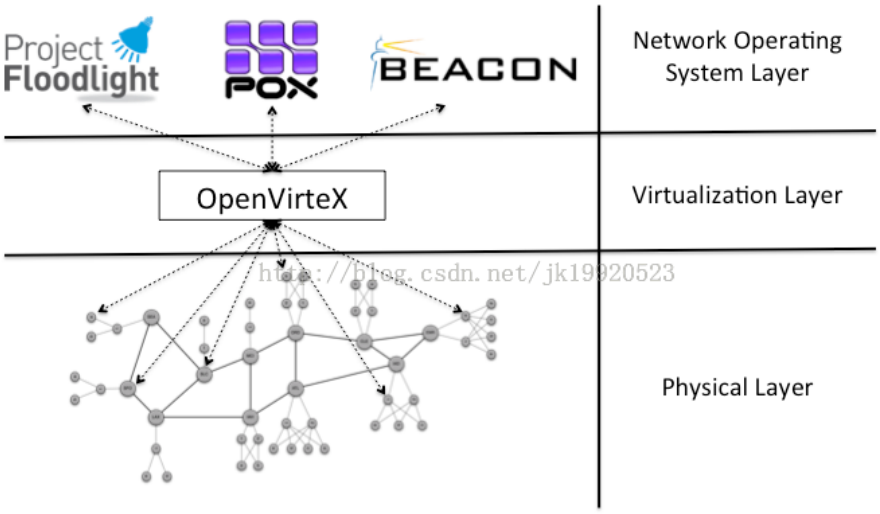

Figure 3: High-Level view of a Virtual Network

It allows you to:

- Create isolated virtual networks with a topology you specify
- Use your own controller
- Use the whole address space

- Change your virtual network at runtime, and automatically recover from physical failures.

SDN Controller: An SDN Controller in a software-defined network (SDN) is the “brains” of the network. It is the strategic control point in the SDN network. We will be using OpenDaylight SDN controller in the project.

OpenStack Neutron: Neutron is part of OpenStack provide 'network as service' between interface devices managed by other OpenStack services (e.g. Nova) it provides a way for organizations to relieve the stress on the network in cloud environment to make it easier to deliver NaaS in the cloud.

If time permits, we would like to use OpenStack Neutron to connect our SDN network with the OpenStack cloud.

4. Acceptance Criteria

The minimum acceptance criterion shall be when we are able to send packets from Host A (connected to switch 1 or 2) located in Network 1 to Host B (connected to switch 1 or 2) in Network 2.

Networks 1 and 2 will be part of separate VMs representing to be part of different administrative domains.

The stretch goals include:

- Solving Design challenges like isolating traffic across different administrative domains, providing value added services like bandwidth guarantee, QoS etc.,
- Solving the administrative problem of mutual agreement (QoS, bandwidth) between multiple vendor
- Exploring the interface between OpenStack Neutron and OpenVirtex (OVX)

5. Release Planning

Detailed user stories and plans are on Trello board

https://trello.com/b/n1veivu7/network-programmability-as-a-cloud-service

- **Release 1 (due by week 5):**

User Story: Creating a simple network for the user

- Creating topology using Mininet

- Using a single SDN Controller like OpenDayLight to send packets across multiple open virtual Switches in Mininet using OpenFlow.

- **Release 2 (due by Week 7):**

User Story: Create a network to represent a single administrative domain

- Introducing multiple mininet networks in the current topology

- Connecting the above created multiple networks to form a larger network within single administrative domain

- **Release 3 (due by Week 9):**

User Story: Introduce OpenVirteX as an abstraction layer to support services across multiple switch and controllers in a single administrative domain.

- Inserting OpenVirteX as a virtualization layer between the controllers and Open Virtual Switches.

- **Release 4 (due by week 11):**

User Story: Introduce a second administrative domain similar to above releases and connect them

- Creating the same design as above using SDN controller in a separate VM.

- Linking the networks by connecting the VMs

References:

1. Networking in Massachusetts Open Cloud (MOC) - Cisco Presentation by Dr. Somaya Arianfar
2. Mininet - https://github.com/mininet
3. OpenStack Neutron - https://www.sdxcentral.com/resources/open-source/what-is-openstack-quantum-neutron/