An Intent-based Network Virtualization Platform for SDN

Yoonseon Han*, Jian Li†, Doan Hoang‡, Jae-Hyoung Yoo§, and James Won-Ki Hong*†

*Division of IT Convergence Engineering, POSTECH, Korea.
seon054@postech.ac.kr
†Department of Computer Science and Engineering, POSTECH, Korea.
{gunine, jwkhong}@postech.ac.kr
‡School of Computing and Communications, University of Technology Sydney (UTS), Australia.
Doan.Hoang@uts.edu.au
§ Ministry of Science, ICT and Future Planning, Korea.
jhyoo@iitp.kr

***Abstract*—Currently, the Software Defined Networking (SDN) paradigm has attracted significant interests from industry and academia as a future network architecture. SDN brings many benefits to network operations and management including programmability, agility, elasticity, and flexibility. With SDN and OpenFlow, one of the promising SDN protocols, software defined Network Virtualization (NV) techniques can be designed and implemented via flow table segmentation to provision independent virtual networks (VNs). In this paper, we propose an intent based virtual network management platform based on software defined NV. The objective of the proposed NV platform is to automate the management and configuration of virtual networks based on high level tenant requirement specifications, called intents. The design and implementation of the platform is based on ONOS, an open-source SDN controller, and OpenVirteX, a network hypervisor. The platform is designed to provide multiple VNs over the same physical infrastructure to multiple tenants. The VNs are isolated from one another allowing tenants to operate and manage their virtual networks independently in terms of network configurations and management policies.**

Keywords*—*Software Defined Networking (SDN), Network Virtualization, Intent Framework, Virtual Network Management and Configuration, Virtual Network Embedding

I. Introduction

Software-Defined Networking (SDN) is a new networking paradigm which enables flexible and efficient network management. The essential principle of SDN is to decouple network control and forwarding functions, and leave each function in its individual network plane. With separated control and forwarding planes, SDN allows a network administrator to program and manage various network elements effectively from the control plane. In the context of SDN, all control related functions are moved to a centralized control plane, hence optimal network control decisions can be made using a global networking view. SDN also provides the ability to simplify network design and operations, with this ability, we can deploy complex network policies (e.g., security governance, access control, accounting, billing) to network infrastructure on the fly. To summarize, SDN brings four major features such as programmability, agility, elasticity, and flexibility to network management domain. By properly utilizing those features, SDN has been promised to reduce CAPEX by using cheap, open, commodity switches and cloud computing for replacing expensive middle boxes and to reduce OPEX by providing simplified and centralized management/operation.

In SDN, the knowledges on managing networks are centralized in control plane such as device and traffic related operation as well as various network services. Network devices in the data plane are only responsible for processing and forwarding ingress/egress packets according to dispatched rules from controllers in the control plane. Communications between control plane and forwarding plane is through the de facto standard OpenFlow protocol. OpenFlow enables the network controller to determine the path of network packets through the OpenFlow enabled switches. The fundamental functionality provided by OpenFlow is remote administration of forwarding tables (flow tables), by adding, modifying and removing packet matching rules and actions. Using this, an SDN controller can collect network information, and deliver proper rules and actions to allow the forwarding plane of network devices. Currently, most of the SDN controllers support OpenFlow as an essential South Bound Interface (SBI) according to the Open Networking Foundation (ONF)'s specifications [1]. ONF has continuously releasing new versions of OpenFlow specification to address various requirements and to promote SDN technologies.

Network Virtualization (NV) is a networking technology that creates dedicated Virtual Networks (VNs) over a physical infrastructure. With the help of NV, multiple tenants are able to share the underlying physical network resources, and they can operate their isolated virtual networks independently. By provisioning VNs over the physical network, network functionality is abstracted from its physical elements. NV technology has the potential to reduce significantly CAPEX and OPEX for network and network service providers with its flexible, on-demand, and scalable provisioning capability. A possible approach to NV is the software-defined (or slice-based) NV whereby a slice of the network physical resources can be allocated to a VN by segmenting OpenFlows flow tables into partitions. Currently, several software defined NV solutions are available including FlowVisor [2], OpenVirteX [3], and FlowN [4]. These approaches do not use tunneling (hence no tunneling overheads), and provide strong Quality of Service

(QoS) and Service Level Agreement (SLA) control compared to overlay NV approaches. Overlay NV approaches refer to NV solutions based on tunneling and encapsulation techniques, and they are usually installed in several datacenter solutions such as VMware NSX [5] and Microsoft Hyper-V [6].

A major deficiency of the software defined NV approaches is that they require underlying network infrastructure to be constructed using OpenFlow protocol. Futhermore, the configuration and management process of each VN are still complicated and time consuming because of the lack of generally available VN embedding methods and automated VN provisioning processes. Currently, to configure and manage VNs, administrators have to deal with all technical aspects of networking such as underlying protocols, addresses, topologies, control rules, and etc. To overcome those complex problems, in this work, we propose an intent-based NV method. The definition of intent is not standardized yet, but it is generally perceived as business or system level policies (or higher level specifications). Intents is independent from specific network technologies and vendor specific features. Moreover, it allows the administrator to use higher level abstraction by using business or system level terminologies and concepts to specify tenants' requirements. With intent, users of tenants only need to concentrate on specifying what they need, rather than how to realize or implement the need. According to ONF's Boulder project working group [7], intent can bring various advantages such as non-prescriptive, portable, universal, technology independent, etc.

In this paper, we propose an intent based VN management platform for SDN to overcome these problems. The fundamental idea is to automatically manage VNs from high-level tenant requirements using intents. To be more specific, the proposed intent-based NV platform addresses the following challenges: 1) using intent to express high-level VN requirement specifications, 2) combining SDN an NV technologies into a single framework, 3) automating the task of VN structure composition and embedding. The platform will host multiple, independent, and isolated VNs, and support multi-tenancy. VNs belonging to different tenants may have different network configurations in terms of network address space, topology, and may be governed by different policies. The proposed platform is implemented on OpenVirtex network hypervisor [3] and ONOS SDN controller [8].

The rest of this paper is organized as follows. Section II summarizes related studies on NV approaches to create and manage VNs. Section III presents the overall architecture of the proposed intent-based NV platform. Section IV describes the concept of intent, and how the platform process intents from the intent specification to installable flow rule generation. Section V describes the implementation details to realize the proposed NV architecture by combining existing SDN and NV softwares. Finally, Section VII concludes this paper with suggestions for future work.

II. Related work

In this section, the current status of NV approaches is summarized. To support NV features, four common NV approaches are compared in terms of characteristics, advantages, and disadvantages. Currently available NV approaches can be classified into four main categories: traditional NV, overlay NV via tunneling, software defined NV, and hybrid NV approaches [9]. In this section, we focus on describing the overlay NV and the software defined NV approaches that can be realized with SDN technologies. Typically, traditional NV approaches refer to VLAN (IEEE 802.1Q) [10] and Virtual Routing and Forwarding (VRF). VLAN allows to partition layer 2 (Ethernet) network into isolated VNs by inserting 12-bit VLAN tag into the Ethernet header. VLAN is a common and simple approach to isolate network traffic into virtually separated networks. However, the maximum number of VNs supported by VLAN is limited to 4,092, and if a sub-network connected though a single port, all hosts in the sub-network should belong to the same VN. VRF is similar to the computing virtualization approach that makes virtual routing instances on a physical routing devices. Each virtual instance is independent, and owns different routing information. But, this technique suffers from complex configuration and management with high CAPEX/OPEX.

Most commercial NV solutions are based on overlay NV approach by leveraging tunneling or encapsulation techniques such as VMware NSX [5] and Microsoft Hyper-V [6]. Overlay NV approach can be further categorized depending on whether the approach supports layer 2 and layer 3 virtualization. Usually, to deliver packets, an ingress network device (switch or router) encapsulates packets by inserting an outer packet header indicating a specific virtual network instance ID (VVID). The encapsulated packets are delivered to the destination according to forwarding rules, and then, decapsulated to restore the original packets at the egress network device. VXLAN [11] and NVGRE [12] are two most representative overlay NV approaches that support layer 2 virtualization, while Generic Routing Encapsulation (GRE) [13] and Locator/Identifier Separation Protocol (LISP) [14] are two most representative approaches that support layer 3 virtualization. The advantages of the approach include 1) only network edges are involved in tunnel encapsulation/decapsulation, and the remainder of the network remains unchanged, 2) theoretically, unlimited number of VNs are supported, 3) VNs are independent from the physical network topology and configuration, 4) VNs mobility can easily be supported. As the overlay NV approach is based on encapsulation mechanism and tunneling, it also brings many disadvantages. The main disadvantages include 1) Two separated networks, VN and PN, are maintained in terms of network services such as management, provisioning, and control, 2) it does not provide mechanisms to provide to guarantee QoS, 3) it introduce high encapsulation and tunneling overheads, and 4) it incurs high management complexity for both VN and PN at the same time. To overcome these disadvantages, cloud platform such as OpenStack provides several tools (such as Neutron) to manage network resources. For SDN, several projects are established to allow OpenStack neutron to communicate with various SDN controllers such as ONOS [8], OpenDaylight [15], and Ryu [16].

With the introduction of SDN and OpenFlow technologies, it is possible to implement NV as an application or a service provided by an SDN controller via flow table segmentation. This software defined NV approach can support layer 1 - 4 network virtualization by matching appropriate packet headers. Performance degradation caused by tunneling overheads is

eliminated. By inserting appropriate forwarding (flow) rules, software defined NV approach can provide specific NV features. Moreover, this approach introduces network abstractions that can be utilized by management application such as virtual links, virtual switches, and virtual routers. Within software defined NV approach, corresponding physical hardwares can be directly programmed for the virtual elements to provide QoS. Tenants can use their VN controller to control their own VNs. Currently, available solutions include FlowVisor [2], OpenVirteX [3], and FlowN [4]. However, the critical disadvantage of the software defined NV is that the physical network has to support SDN and OpenFlow. The hybrid NV approach refers to NV solutions combining these NV approaches to create and manage VNs.

III. Overall Platform Architecture

The proposed intent-based VN management platform is designed to have a hierarchical architecture consisting of multiple layers to apply the Separation of Concerns (SoC) design principle. The platform plays two roles which are 1) network hypervisor and 2) SDN controller. As a network hypervisor, the platform posesses VN management capabilities such as VN provisioning, modification, and removal, at the same time, it also provides interfaces to relaying VN events and messages to external entities running tenant specific applications. As an SDN controller, the platform provides network abstractions and control capabilities for both physical and virtual networks. The overall architecture is depicted on Fig. 1. The design objectives are 1) to support multi-tenancy, 2) to provide network abstractions for application development, 3) to allow high-level tenant requirements specification using intents, and 4) to support tenant specific controllers and applications by providing various interfaces. The platform has five layers: protocol adaptation, abstraction, virtualization, virtual abstraction, and intent layer. The platform design is inspired by the architecture designs of ONOS [8] and OpenVirtex [3].

The protocol adaptation layer is responsible for processing protocol specific messages which are used to communicate with physical hardwares such as OpenFlow switches. The main roles of this layer are: 1) decoding protocol specific messages and delivering them to proper providers located in the abstraction layer, 2) managing communication channel between the controller and network devices, and 3) receiving the requests from upper layers, encoding the request into protocol specific messages and transmitting to hardware devices through communication channel.

The abstraction layer is responsible for abstracting protocol specific concepts and hiding the details of underlying infrastructure from upper layers. The abstractions can represent various elements and properties in a protocol-agnostic manner. Some representative abstractions are Device, Link, Topology, Event, Path, Flow, etc. There are several abstract components residing in this layer, and each abstract component is in charge of abstracting the network concept for each protocol.

The main responsibility of the virtualization layer is to translate VN objects into physical objects by maintaining mapping information. The mapping information includes address mapping and topology mapping. In the simplest case, virtualization layer simply replaces the address and topology

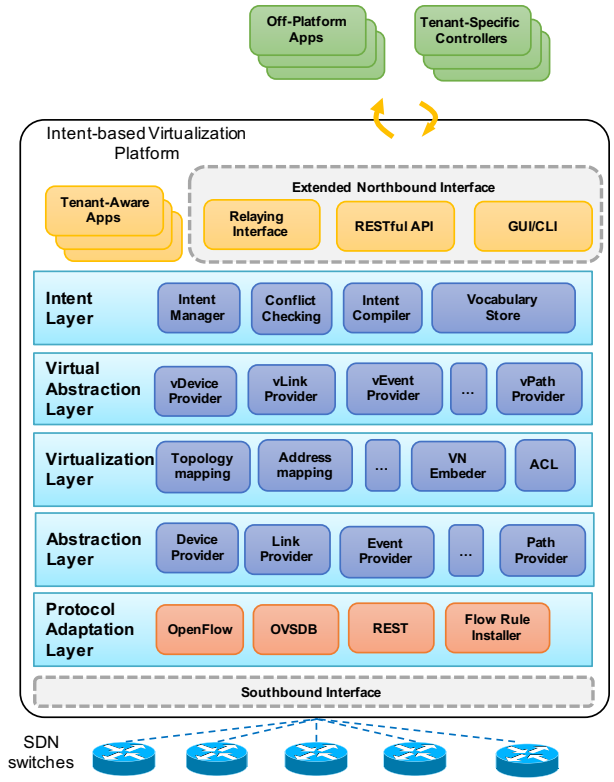

Fig. 1. The overall platform architecture design

information of virtual resource into that of physical resource. To support various strategies to satisfy different user requirements, the platform design should consider to address multiple embedding algorithms as plug-ins. In the platform design, a *VN embedder* plays the role of a matchmaker between resources and VN embedding algorithms. Moreover, the actual embedding algorithms can be deployed as an off-platform component because it requires large computing power, hence not suitable to be part of an on-platform component. The reasons behind this design decision is discussed in the Section VI.

The virtual abstraction layer provides network abstractions for tenant VNs. The fundamental difference between virtual networks model objects and physical network model objects is that virtual objects can be freely created and removed on the top of physical objects. However, physical objects have strong dependency on physical network infrastructure in terms of protocol, topology, network addresses, and links. The usage to consume virtual objects is same as the physical objects provided by lower abstraction layer after the creation of the objects.

The intent layer allows tenants to specify their high level requirements independent from low level details such as MAC address, IP address, topologies, etc. The services, which are located in the intent layer, provide 1) intent objects consumed by applications; 2) an intent confliction checking service between different intents to avoid conflicted and illegal intents; and 3) an interface to the administrator to feed the information that is used to interpret and translate the intents into installable flow rules. For example, “Database” specified in an intent should

be translated into network address "192.168.0.1" with port number "3306". The stored information can specify various entities such as human domain exerts and network management protocols provided by the protocol adaptation layer. The details about intent are described in following section. The intent layer provides extended North Bound Interface (NBI) that is consumed by various applications.

According to location where the applications are executed, they can be categorized into two types which are on-platform and off-platform applications. On-platform applications are developed with tenant-awareness using the abstractions provided by the virtual abstraction layer. Therefore, they can be shared by multiple tenants with different VN views. For example, visualization applications can be consumed by different tenants to show graphs or topologies for their own VNs. To support off-platform applications, a special application, called relaying interface, is provided. The responsibility of this application is to deliver messages or events from the virtual abstraction layer to external entities. This application enables the communication to tenant specific controllers and applications similar to that in a traditional network hypervisor such as FlowVisor [2]. However, off-platform controllers and applications are not aware of the existence of other tenants.

IV. Intent Based VN Management and Configuration

In this section, the definition of "intent" for the proposed platform is described. In doing so, we define components to translate high-level intents into installable flow rules. We also define the lifecycle of intents in order to efficiently manage them.

A. Definition of Intent

Oxford dictionary defines "intent" as "an aim or plan or purpose." The definition of intent for network management is commonly perceived as as business or system level policies specified with common concepts and terminologies agreed by all related stake-holders such as business managers, application developers, and network administrators. However, a clear and concise definition of intent for network management has not been standardized yet. Several projects are proposed to introduce intent for SDN application development and network management based on high-level requirements or management policies. Recently, intent-based interface has been pursued rigorously by major open-source project communities (ONF, ONOS and OpenDaylight) to provide a standardized intent-based northbound interface for SDN [7], [8], [15].

By using intent based interface to specify high-level requirements, consumers (e.g. applications developers) can program network services without concerns for technical specifics and implementation details. Intent tends to be more concentrating on describing the outcome rather than the process that dictates the decisions toward the outcome. By summary, intent is used to describe "what" the user want, but not "how" to realize it (with respect to resources, constraints, and actions). From the users perspective, technology-agnostic interfaces are more desirable. The intent based interface shields the complexity of underlying networks and allows users to focus on expressing their network service demands.

In this paper, an intent is defined as a high-level policy specified in common concepts and terminologies, and interpretable by both tenants (network service consumers) and network service providers. However, this does not mean that all policies are specified in a business-level or system level terminologies. In that manner, our platform will provide an extensible method for defining supported concepts and terminologies to support future applications. Our objective introducing the concept of intent is to mitigate the network management obstacles. With intent, we expect that the knowledges that are required for managing network is significantly reduced in application developers' perspective.

B. Intent Specification for VN Management

The first step of performing intent based management is to provide an interface to express high-level specification as a form of intent. There are several ways to specify high-level specification. The specification can be either defined by using language (e.g., NEMO [17]) or graph (e.g., PGA [18]). We can also make use of specific intent APIs to describe the specification such as intent APIs that are defined in ONOS and in work [19]. The way that is provided by the proposed platform is based on an intent framework, with which tenant applications can consume intent model objects. The underlying design of the proposed intent framework is inspired by the ONOS's intent framework. The pre-defined types of intent objects, which could be extended to address tenant specific intent types, are depicted in Fig. 2. The proposed platform also provides high level network abstraction model objects (e.g., host, switches, link, and middle-boxes).

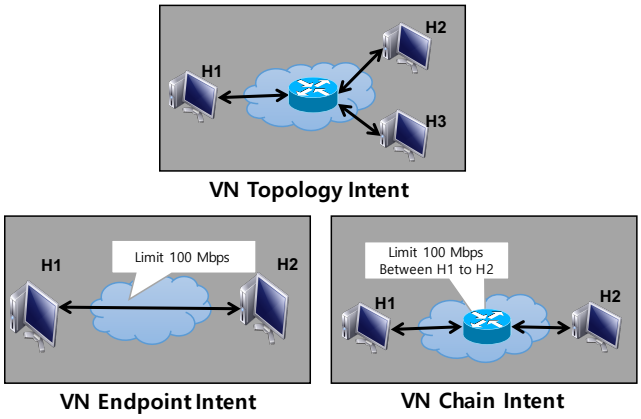

Fig. 2. Three basic types of intents provided by the proposed platform

- VN Topology Intent: This type of intent only expresses the connectivity relationships among network nodes (i.e., hosts and virtual switches) without specifying VN behaviors. The network behaviors such as packet forwarding or management policies are managed by SDN controllers.

- VN Endpoint Intent: This intent allows to express high-level requirements for tenants endpoints without concerns of supporting network infrastructure. Because only endpoints are involved in the intent specification, a tenant application developer can entirely focus on describing the relationship between endpoints. Various relations between endpoints such as 1 to 1, 1 to many, and many to many, are possible.

- VN Chain Intent: This intent type is an extended intent type from the *VN Endpoint Intent* to chain intermediate network behaviors. This type of intent expresses network service chains by composing virtual network functions or physical middle-boxes between source and destination endpoints.

To specify intents accurately, we need to define the context that describes what, when, and how the specified intents should be applied. To express a context, an intent object requires four attributes; resources, conditions, priority, and instructions. **Resources** refers to a set of virtual network objects involved in intent specification. **Conditions** are a set of criteria that describes when the intent will be activated. The criteria can include header matching field and network conditions such as Maximum Link Utilization (MLU). **Priority** is used to determine the execution order of intents. **Instructions** refers to a set of actions that to be applied to the packets which satisfy resources and conditions what have been defined. To support **Instructions**, the platform provides a way to abstract network behavior such as forwarding, filtering, and drop.

An alternative way to specify intents is through networking graph. Describing requirements in graph is the most intuitive way to go with, and through this way, tenants are able to specify their own requirements step-by-step rather than whole network structure. Designing the whole network structure is complex and error-prone task. This approach is inspired by PGA [18].

C. Intent Composition and Conflict Checking

After specifying intents for each tenant requirement, the next step is to aggregate all intents to construct the requested VN which consists of a set of network objects and behavior abstractions. The intent composition process is required to translate high level specifications into a network driven concepts and terminologies. First of all, we need to identify and manage VN endpoints. This is needed to unify and translate concepts, resources, and terminologies into a single unified form agreed by all involved entities. To address this issue, [19] proposed an end-point discovery protocol, while [18] used "label" to apply policies. Note that "label" contains a group of pre-defined endpoints with the same set of policies. The composition process is a step-by-step aggregation process of intents into a network graph model consisting of virtual objects. As outputs of intent composition, VN topology, address space and policies that are used for governing the VN are computed. The overall intent composition process is described in Fig. 3.

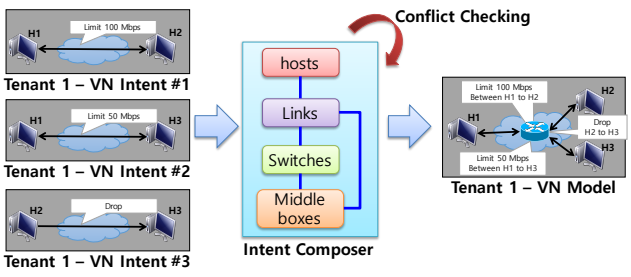

Fig. 3. Intent composition and conflict checking process

During the composition process, the proposed platform can verify incorrect intents and detect conflicts between intents. By inspecting intent resources, conditions, priorities, and instructions, an intent composer can detect conflicts. First of all, we need to identify the relationship between intents to check whether they have any dependency on each other. If any of two intents are interdependent, the platform will lookup the priorities of intent to check whether the instructions are mutually inclusive. For instance, if two intents share the same pair of source and destination addresses, and the identical instructions are defined in each intent, then they are detected as conflicted intents (possibly through duplication). The intents that encounter any conflict required to be modified and negotiated into composable intents. At this stage, this is beyond the scope of this work. We will discuss this in a separate research work.

D. Intent Mapping and Installation

Our network representation consists of a set of network model objects and network behavior abstractions. To embed a VN into existing physical network resources, virtual model objects should be bound to actual physical objects. At the meanwhile, the network behaviors also require to be translated from virtual network behaviors into installable physical network behaviors. A physical network provides several ways to accommodate a VN model, and the ways of accommodation are determined by VN embedding algorithm. The objective of VN embedding algorithm is to find an optimal mapping between the VN and physical resources with considering to satisfy a set of requirements defined by network administrator. The design of VN embedding algorithm is beyond the scope of this paper. To efficiently embed VNs according to management objectives, the platform can adopt various algorithms introduced in [20].

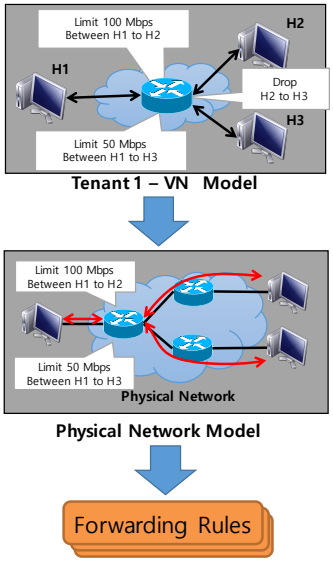

Fig. 4. Mapping process from VN model to installable flow rules

To translate VN model objects into physical objects, the platfrom should bind all virtual entities into concrete network nodes. This process can be supported by managing tenant's end-points. Discovering and managing end-points of VNs are

challenging because the terminologies are not standardized among stake-holders yet. To be a solution, in this paper, we introduce a concept of "vocabulary store" which refers to a knowledge store that contains information from various sources include 1) human domain experts and/or network administrators, 2) host and network discovery and 3) management protocols. In this case, we need a method to represent the relationship between entities used in intent and low-level details. To efficiently querying the equivalence between entities, a ontology based knowledge representation is adopted for the platform to support a semantic based inference mechanism. This does not mean that "vocabulary store" simply generates new knowledge by using inference rules, but it supports checking and finding simple equivalent relationships between entities.

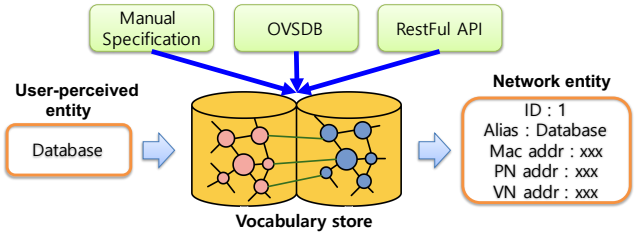

Fig. 5. The role of vocabulary store

E. Intent Lifecycle Management

To manage intents specified from multiple tenants, we have designed a Finite State Machine (FSM) that represents the state of each submitted intent. The FSM traces the entire lifecycle of each intent from intent submission to intent withdrawal. The main advantage of the state lifecycle management is that it allows the platform to determine wether the tenant requirements are satisfied based on their corresponding VN's status. Moreover, a recovery plan can be made if an abnormal state of intent is detected due to hardware failure or attack. Fig. 6 depicts the state transition diagram.

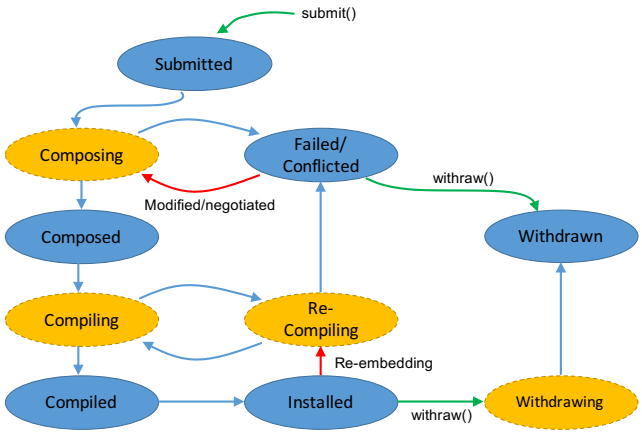

Fig. 6. A finite state machine to represent the intent lifecycle

V. Implementation

In this section, we describe implementation details for realizing the proposed platform design mentioned in previous section. To accelerate our software development and make our contributions be available in public, we decided to adopt open source. To the best of our knowledge, OpenVirtex [3] and ONOS [8] are most suitable opensources in terms of reusability. We adopted those two open sources as base software stack.

A. ONOS and OpenVirtex

The high-level design of the proposed platform is implemented on top of OpenVirtex and ONOS. In the proposed platform, OpenVirteX is used to realize NV features including VN topology, addresses, and configurations, while ONOS is used to provide SDN control functions and network abstractions consumed by tenant applications. Before we go in details on our implementation, we first explain the architecture and design principles of ONOS and OpenVirtex.

OpenVirtex is an open source network hypervisor based on the software defined NV approach with OpenFlow semantics. OpenVirtex achieved NV as a proxy between the physical network and the tenants SDN controllers. In the view of proxy architecture, OpenVirtex is similar to FlowVisor [2]. However, OpenVirtex provides advanced NV features such as supporting arbitrary VN topology and addressing schemes, and configuring VN as per tenant request. All OpenFlow messages generated by each tenant controller are translated into the corresponding messages by OpenVirtex, and eventually transmitted to physical fabrics. By supporting OpenFlow natively, this approach delivers programmable VNs to tenants. In other words, OpenVirteX acts as a (de)multiplexer for OpenFlow messages between the multiple VN control planes and the physical network infrastructure. OpenVirteX provides two types of APIs, monitoring API and tenant API. The monitoring API is a read only interface used to inquiry VN configurations and state information, while the tenant API is used to create and configure VNs. Both APIs use JSON-RPC protocol to exchange information.

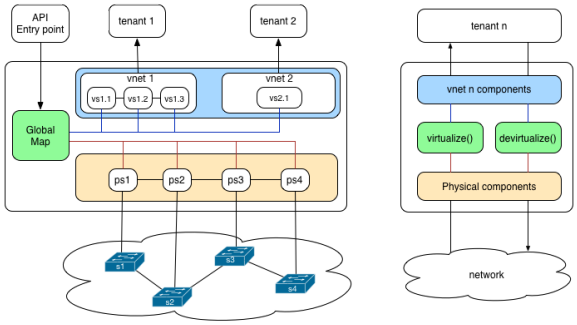

Fig. 7. The OpenVirtex architecture [3]

ONOS adopted layered architecture, and each layer is charging of a set of responsibilities as shown in Fig. 8. To stitch each layer, ONOS provides North Bound APIs (NBIs) and South Bound APIs (SBIs). NBIs are used to communicate with on/off platform applications with higher level of abstractions, while SBIs are used to communicate with network devices using specific protocols. The protocol layer is in charging of encoding/decoding protocol specific messages. The provider layer abstracts protocol specifics into more general concept and communicates with core layer through SBIs. The main responsibility of core layer is to provide: 1) abstracted data

model, 2) message relaying between application and provider layers, and 3) distributed primitives.

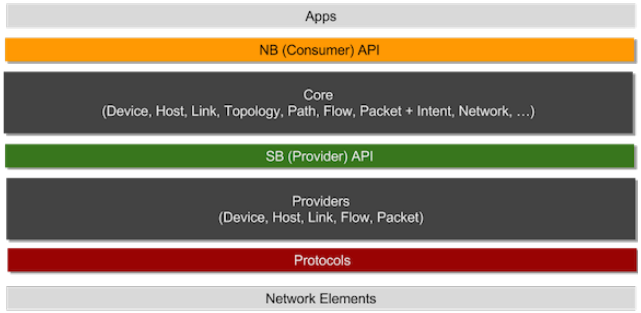

Fig. 8. The layers of ONOS architecture [8]

B. Virtualizaiton Subsystem

To integrate ONOS and OpenVirtex, we extended SBIs and developed a dedicated provider for OpenVirtex. The roles of OpenVirtex provider are: 1) to manage communication channel between ONOS and OpenVirtex; and 2) to translate requests generated from OpenVirtex into ONOS consumable format. Note that all information is exchanged using JSON-RPC format, and to support conversion between JSON-RPC messages and network objects, we developed a component that serialize/deserialize virtual network objects into a documents for device, link, switch, topology, and etc. Moreover, we defined a VN manager component that supports OpenVirteX management operations within ONOS. Fig. 9 shows the chain of OpenVirteX and ONOS to realize the proposed VN platform.

The fundamental SBI of the platform is the OpenFlow protocol between the tenant control planes and the physical network infrastructure. Using this protocol, it is possible to deliver OpenFlow messages generated by the physical OpenFlow devices to each tenant VN control plane. In this case, all VN messages are delivered through the shared OpenFlow channel for all tenant VNs. To identify destination VN of each OpenFlow message, it is necessary to maintain mapping information.

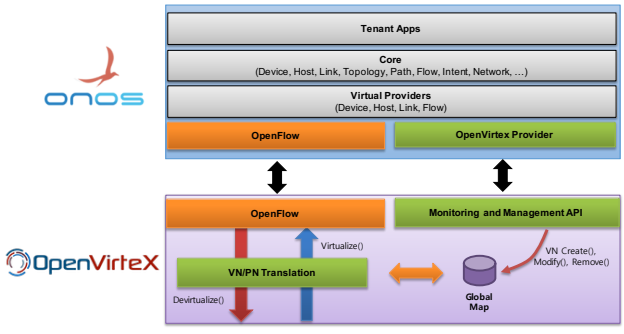

Fig. 9. The implementation of the proposed platform on the top of OpenVirteX and ONOS

C. Virtualization and Virtual Abstraction Layers

ONOS provides a rich set of network abstractions in terms of network model objects such as device, link, port, flow, and etc. However, ONOS and those models are originally designed for managing a single SDN network, rather than multiple VNs. To support multiple VNs, we have implemented VN model objects by extending existing object model. The fundamental differences between the VN model objects and existing model objects are 1) the addition of attributes needed to identify tenants, 2) the target of operations. The operations on virtual objects must be translated into the operations on physical objects to be installed on the physical devices by referring to mapping information.

To help translation process from virtual objects to physical network model objects, we also need to abstract network operations as consumable objects such as forwarding, filtering, and drop. These behavioral abstractions make the translation process to be simple and transparent. Fortunately, ONOS has already provided flow abstraction model named "Flow Objective". Originally, "Flow Objective" was proposed to hide the forwarding behavior of OpenFlow devices, as different OpenFlow devices have different pipeline implementation. Flow Objectives allows to describe a SDN applications objective rather than hardware specific implementations. To take advantages of flow objectives, our platform manages all network behaviors as flow objectives from virtual network model to physical network model.

D. Intent Layer

To implement intent components inside the intent layer, we extended ONOS's intent framework. The goal of the ONOS intent framework is similar to our intent based VN management because it allows applications to specify their network control desires in a form of policy rather than mechanisms. However, there is a fundamental difference between the intents for physical network management and the intents for VN management. In the physical management case, intents are used to make high-level requests to consume the existing network model objects. However, for VN management, the platform has to create virtual network model objects, not consume. For example, from an intent that desires a connection between host A and host B, the platform has to create virtual objects representing virtual switches, links, and ports. To address this problem, we extended the existing ONOS intent framework to include the capabilities to manage virtual objects and operations. We reused the classes to be used for intent specification such as *Host to Host intent*, but the entire process for handling intents is re-redesigned and implemented suitable to the VN management process.

E. Access Control and Applications

Two types of applications are supported by the platform; on-platform applications and off-platform applications. For the off-platform applications, the platform just needs to relay OpenFlow messages to external entities. However, supporting on-platform applications is challenging because on-platform applications may be shared by multiple tenants. For example, a routing application should be adopted to support multiple VNs having different network topologies and addresses. To overcome this problem, we re-designed ONOS provider interfaces to make an application as a "blackbox". In our platform, an application is a closed system that just provides results in response to it's input. Therefore, the state of each

VNs used as applications's input should be managed outside the application domain. To support this design paradigm, we have introduced a "VN context store" that stores tenant specific information about internal and/or intermediate data consumed by an application. By switching the context store, an application can be shared by different tenants.

One of the most challenging issues in realizing a VN management platform is to isolate VNs from different tenants. An unauthorized access to virtual objects or resources from an application that is not belonging to the owner tenant would cause wrong operations and security concerns. To manage access privileges of each tenant, we have to develop an access control feature. To realize the feature, we implemented a component that intercepts messages between applications and ONOS providers to ensure that they all belong to a same tenant, otherwise the messages will be dropped with error reprots. The implementation is similar to an ONOS subsystem, called security-mode ONOS, that provides application authentication and access control services for ONOS northbound APIs.

VI. Discussion

Currently, the proposed VN platfrom is still under the development. This section introduces further issues for consideration.

A. Native Support of Virtualization Layer

In this paper, we described the initial design and implementation of the proposed NV platform. To accelerate the development process, we decided to reuse two separated opensources, OpenVirteX and ONOS. However, this initial implementation approach does not produce optimal design to elicit high performance. A performance bottle neck may be introduced with the opening of OpenVirtex and ONOS to external interfaces, as this requires heavy workload to translate ONOS abstractions into JSON documents used in OpenVirteX API. To overcome this problem, we need to migrate OpenVirteX functions to support virtualization layer in ONOS natively.

B. Optimal Virtual Network Embedding

Optimal mapping between virtual and physical objects may be different depending on the objectives of the VN consumers and providers. For example, low energy objective will include the minimum number of physical devices, but high performance objective will not. There exist VN embedding algorithms for map virtual and physical network model objects, under specific performance criteria. To address this issue, we consider supporting a strategy pattern that allows the selection of the most suitable VN embedding algorithm for the target application.

C. Virtual Network Migration

The initial optimal mapping between between virtual and physical objects, however, my no longer valid due to VN creations and removals, and physical network infrastructure failures. In addition, due to changes in the environment, a running VN may need to migrate, The platform toned to reflect the new optimal mapping. In this migration process, all VN states must be managed in a way that conserves VN structure and configurations to prevent loss of information. Moreover, a method is needed to reduce service down time to a minimum. It means that VN mobility should be supported in a similar manner to VM live migration or vertical handover. This implies the need for representing internal VNs state/configuration information in order to migrate them efficiently.

VII. Conclusion

In this paper, we have presented an intent based VN management platform. The design objective of the platform is to automate VN management process based on intent that allows expressing high-level tenant requirements. To realize the objectives, we have described the high level design and implementation approach. The proposed platform is based on a hierarchical architecture consisting of five layers to isolate specific level of concerns from high-level requirements to installable flow rules. The proposed platform can bring several advantages, 1) an integrated NV platform taht integrates seamlessly the network hypervisor and the SDN controller, 2) an intent-based management platform that allows management applications to express their need in management domain specific language and the management providers the freedom to implement the required service, and 3) an automated VN management method can be developed from high-level intents.

The proposed VN platform is still under development. For future work, the top priority task is to finish the development according to the described design. Furthermore, our design will be refined to address the features mentioned in the Discussion section. Once the implementation is completed, a comprehensive evaluation of the functionalities and the performance of the platform will be presented with several use cases. We also plan to publish the platform as an opensource software.

References

[1] Open Networking Foundation., *OpenFlow Switch Specification Version 1.0.0*, Std., Dec. 31, 2009.

[2] R. Sherwood *et al.*, "Flowvisor: A network virtualization layer," Tech. Rep., 2009.

[3] Al-Shabibi *et al.*, "Openvirtex: Make your virtual sdns programmable," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 25–30.

[4] D. Drutskoy *et al.*, "Scalable network virtualization in software-defined networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, 2013.

[5] VMware NSX, "The platform for network virtualization." [Online]. Available: https://www:vmware:com/files/pdf/products/nsx/VMware-NSX-Datasheet:pdf

[6] A. Velte and T. Velte, *Microsoft Virtualization with Hyper-V*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2010.

[7] Open Networking Foundation., "Project boulder: Intent northbound interface (nbi)." [Online]. Available: http://opensourcesdn.org/projects/project-boulder-intent-northbound-interface-nbi/

[8] P. Berde *et al.*, "Onos: Towards an open, distributed sdn os," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 1–6.

[9] M. Jim and M. Ashton, "The 2013 guide to network virtualization and sdn," Dec 2013.

[10] *IEEE 802.1q: VLAN*, IEEE, http://www.ieee802.org/1/pages/802.1Q.html, 2005.

[11] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, "Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks," RFC 7348, August 2014.

[12] M. Sridharan, K. Duda, I. Ganga, A. Greenberg, G. Lin, M. Pearson, P. Thaler, C. Tumuluri, N. Venkataramiah, and Y. Wang, "Nvgre: Network virtualization using generic routing encapsulation," Internet Draft, September 2011.

[13] S. Hanks, D. Meyer, D. Farinacci, and P. Traina, "Generic routing encapsulation (gre)," 2000.

[14] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "Rfc 6830: The locator/id separation protocol (lisp)," 2013.

[15] The OpenDaylight Project, Inc., "OpenDaylight - Technical Overview," 2013. [Online]. Available: http://www.opendaylight.org/project/technical-overview

[16] Nippon Telegraph and Telephone Corporation, "Ryu Network Operating System," 2012. [Online]. Available: http://osrg.github.com/ryu/

[17] Y. Zhang, Y. Xia, S. Jiang, T. Zhou, and S. Hares, "NEMO (NEtwork MOdeling) Language," Internet Engineering Task Force, Internet-Draft draft-xia-sdnrg-nemo-language-04, Apr. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-xia-sdnrg-nemo-language-04

[18] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang, "Pga: Using graphs to express and automatically reconcile network policies," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/2785956.2787506

[19] R. Cohen, K. Barabash, B. Rochwerger, L. Schour, D. Crisan, R. Birke, C. Minkenberg, M. Gusat, R. Recio, and V. Jain, "An intent-based approach for network virtualization," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, May 2013, pp. 42–50.

[20] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 1888–1906, Fourth 2013.