HyperExchange: A Protocol-Agnostic Exchange Fabric Enabling Peering of Virtual Networks

Saeed Arezoumand, Hadi Bannazadeh, and Alberto Leon-Garcia
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON, Canada
Email: s.arezoumand@mail.utoronto.ca, hadi.bannazadeh@utoronto.ca, alberto.leongarcia@utoronto.ca

***Abstract*—With the growing pervasiveness of virtualization technologies, carrier networks are shifting from simple packet delivery platforms to multi-tenant integrated clouds offering fine-grained resource management. The need for interoperability among these autonomous cloud-based service providers has created demand for versatile and extensible exchange points to interconnect the future Internet. A novel SDX (Software Defined Exchange) can address this challenge and help redefine the Internet exchange by leveraging SDN. Current implementations of SDXs have focused on traffic exchange between conventional IP networks and have not been specifically intended for exchange between multi-tenant environments and virtual networks; and they have mostly relied on OpenFlow for network forwarding and functionality. While OpenFlow is the de-facto solution for fine-grained forwarding, it nevertheless provides limited network functionality. In this paper we present HyperExchange, a protocol-agnostic exchange fabric for peering of virtual networks. HyperExchange is designed to provide exchange services between autonomous Infrastructure Providers and their hosted Virtual Networks. As a result, it specifically offers solutions for inter-domain tenant authentication and authorization for network control. By leveraging SDI as the core building architecture, HyperExchange uses SDN to forward and steer traffic in a fine-grained manner and yet relies on NFV to push all network functionalities to standard servers as software-based functions. This solution meets both scalability and extensibility requirements for long-term use. We have deployed a prototype of the HyperExchange between SAVI and GENI testbeds to serve real world exchange experiments.**

***Keywords*—Internet Exchange Point, Software Defined Exchange, Network Virtualization, Software Defined Infrastructure**

I. Introduction

A. Background

Current inter-domain networking has evolved based on a standard structure: A group of Autonomous Systems (AS) using IP as the internal network structure while relying on BGP for inter-domain peering. This structure made inter-networking possible in the first place, but afterwards it led to a notorious ossification. The problem, more precisely, is rooted in two major sources: First, the existing all-IP foundation of ASs forces a fixed addressing scheme which is location-based and assigned through standard address registries. Moreover, routing based solely on destination IP prefixes results in a severe control inflexibility in internal operation of ASs.

The other major challenge of the current Internet arises where different ASs interconnect with each other at exchange points. Use of BGP as the basis of inter-domain networking has caused a set of unresolved issues[1] including:

- Difficult and painful troubleshooting and security maintenance,
- Large convergence times,
- Anomalies caused by possible routing inconsistencies,
- Adversity of QoS policy expression and enforcement,
- Unoptimized end-to-end paths due to triangle inequality violations.

While these challenges can be traced back to the technology limitations in the early years of the Internet, recent advancements in SDN and NFV have provided new capabilities to motivate a reconsideration of inter-domain networking beyond its conventional restrictions. Along these lines, two major trends are ongoing in the research community and industry. In one direction, Software Defined Exchanges (SDX) [2] have been introduced to make Internet Exchange Points (IXP) [3] more flexible and to facilitate inter-domain routing while keeping ASs with the traditional all-IP structure. In the other direction, several attempts has been made to redefine the network foundation of autonomous carrier networks and Internet Service Providers (ISPs) towards integrated and multi-tenant clouds and datacenters offering programmable and fine-grained virtual networks [4][5][6]. In this new model, the traditional role of ISPs will be segregated into two roles: Infrastructure Providers (InPs), who provide virtualizable network infrastructure and Service Providers (i.e. tenants) who use virtual networks to provide services for end-users [7]. SAVI [8] and GENI [9] testbeds are two real deployments of such InPs. However, flexible peering of InPs and their hosted VNs has remained a challenge.

B. Requirement Analysis

The above mentioned trends points towards realization of anInternet of Virtual Networks (IVN). An ideal exchange point for this IVN must provide the following capabilities:

Protocol Agnosticity: A central feature of any Virtual Network Environment (VNE) is the customizability of network protocol and logic [7]. As a result, the exchange point must provide exchange services for different types of networks independent of the protocol being used.

Extensibility and Flexibility of Peering: In order to provide on-demand peering and traffic exchange services,

978-3-901882-89-0 @2017 IFIP

exchange points must enable rich functionalities on network traffic that can range from a simple modification of header values to a complex stateful Deep Packet Inspection system. OpenFlow [10] protocol is a proper solution for fine-grained traffic forwarding. However, relying on a hardware switch as the only packet processing pipeline will narrow down the network functionalities of exchange point to a limited set of header modifications. Thus, software-based packet processing frameworks, such as P4 [11] or DPDK, are needed to overcome OpenFlow limitations. To this end, the exchange point architecture must include processing resources in addition to pure networking resources.

Multi-tenancy: Policy enforcement to the exchange point should not be limited to InP's. Tenants of InP's (i.e. owners of VN's) should have the ability to define their desired exchange policies so that end-to-end orchestration over VN's will be possible. An important requirement to realize this feature is a well-defined network flow space authorization at exchange point that can isolate incoming or outgoing traffic of VN's against each other.

Scalability: Large scale IXPs can have over hundreds of participants each of which can have hundreds of thousands of prefixes [3]. Each of these participants may define different policies for network flows, and the fact that these participating networks can have multiple tenants, introduces scalability requirements for both data and control planes.

C. Overview

To address the aforementioned trends, we have introduced the concept of HyperExchange as an exchange fabric for InP's and their hosted VN's. HyperExchange, in summary, provides the following particular contributions:

- A formal model that redefines the conventional concepts of network domain and sub-nets and a uniform data-model for networks and their sub-nets. We have extended the formal model to specify the pipeline of HyperExchange formally (Section 2)
- An extensible design for data-plane pipeline that leverages OpenFlow and custom VNF's to provide arbitrary packet processing capabilities. (Section 3)
- A control-plane design based on SDI model to provide multi-tenancy and to enable tenants of InPs to enforce network control policies on their own slice of traffic (Section 4)

Our proposal for HyperExchange has gone beyond a paper design. We have implemented and deployed a proof of concept implementation between SAVI and GENI testbeds. Our efforts towards prototyping HyperExchange and use-case experiments is discussed in Section 5. In Section 6 we will review related work and we will conclude our work and discuss the future work in Section 7.

II. FORMAL SPECIFICATIONS

The network model at an exchange point specifies a clear semantic to bind slices of traffic (incoming or outgoing) to each participating network. This binding is central to define any traffic management in the exchange point. Figure 2

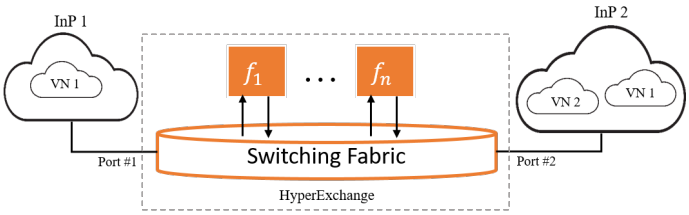

Fig. 1. Conceptual representation of HyperExchange

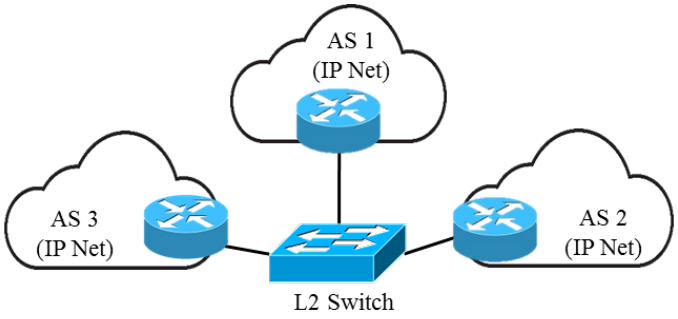

Fig. 2. A conventional IXP peering three AS's

demonstrates a sample IXP and three participating ASs. In this case the IXP is modeled as a Layer 2 switching fabric and each participating network is connected to the IXP via a physical connection. Since all of the participating networks are IP networks, the mapping between networks and traffic at IXP is defined by source and destination IP addresses. This model forms a simple two-dimensional flow space in which source and destination IP addresses are dimensions. Figure 3 represents the slice of traffic coming from AS1 and going to AS2. While this model greatly simplifies the traffic slicing in IXP, it enforces two main limitations for inter-networking. First, only public IP networks can participate in an exchange point and second, forwarding logic at IXP is mainly defined by IP prefixes. Neither conventional IXPs nor current implementations of SDXs have targeted exchange of traffic between VNs with customized network protocols. Depending on vendor specific features, some exchange points may offer

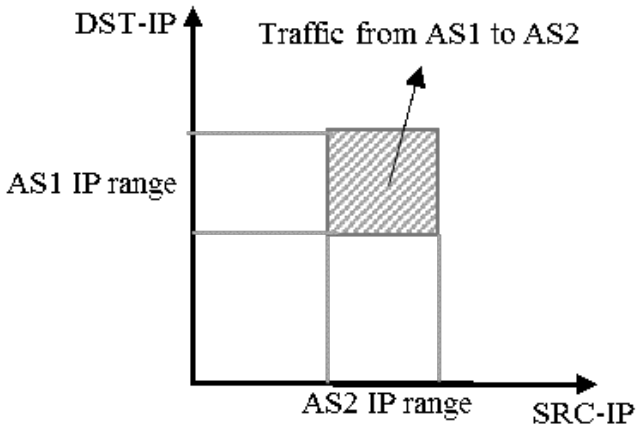

Fig. 3. Two-dimensional flow space of current exchange points

limited support of non-IP protocols but no exchange point have been introduced with a protocol-agnostic model that satisfies VN peering requirements mentioned in the previous section.

A. Geometric Model

The network model of HyperExchange is built on a geometric model which is inspired by HSA (Header Space Analysis)[12] with some modifications. The perspective in HSA is a network with a set of switching boxes and their interconnecting links; while HyperExchange model is defined for traffic at a single point (exchange point) that interconnects arbitrary InPs and VNs. As a result, topologies or protocols of participating networks are not important and a network at exchange point is specified by the set of packets that belongs to it. In this model, each packet is considered as a point in a geometric space. Using this formalism, a clear specification for networks and subnets are provided. The model is then extended to define InP and tenant control policies and the main pipeline of HyperExchange.

1) Basic concepts: We Start with a brief introduction of the basic spaces and concepts of our model.

Total Header Space: A header space of H with length of L can be represented by $\{1,0\}^{L}$ [12]. For example, the header space of IP address header can be defined by $\{1,0\}^{32}$. A header field can be any bit sequence in the packet including the content. Consider the set of headers $\{\mathrm{h}_1,\ \ldots\ \mathrm{h}_n\}$ with length of $\{\mathrm{L}_1,\ \ldots\ \mathrm{L}_n\}$ as list of interested fields; the total header space is the cross product of all header spaces defined by this list:

$$\mathbb{H} = \ \times_{\mathrm{i}=1}^{\mathrm{n}}\{0,1\}^{\mathrm{L_i}} \tag{1}$$

Port Space: A space represented by the total set of interconnecting ports of exchange point. These ports can be physical or logical ports. In order to preserve the generality of the model, a packet drop can be modeled by sending the packet to a logical port associated with drop.

$$\mathbb{P} = \{\mathrm{P}_1,\ \ldots,\ \mathrm{P_m}\} \tag{2}$$

Flow Space: The flow space is the cross product of $\mathbb{H}$ and $\mathbb{P}$. Each packet in an exchange point belongs to two networks: source and the destination. To solve this conflict, we exclude incoming flow space and outgoing flow space at exchange point. To distinguish these spaces, we use a single bit binary vector. Thus the total flow space of HyperExchange will be:

$$\mathbb{F} = \mathbb{H} \times \mathbb{P} \times \{0,1\}^{1} \tag{3}$$

Match Expression: A Boolean expression defined over header values and/or port numbers. For those headers that have source and destination values such as IP and MAC, the source value will be matched in incoming space while the destination value will be used to match in outgoing space. The same principle is applied for incoming and outgoing port numbers. Each match expression is associated with a region in the flow space which is the basic building block of flows. A flow is represented by "F" and formally is a subset of the flow space.

Filter Function: Given a set of packets, a Match Expression M, returns a subset of packets which constitutes the flow defined by M. The behavior of the Filter Function for a single packet is:

$$\Psi_M\left(\{pkt\}\right) = \begin{cases} \{pkt\} & if\ M\ holds\ for\ pkt \\ \phi & otherwise; \end{cases} \tag{4}$$

The output of the Filter Function on the entire Flow Space is the flow associated with M, that is the region defined by M in $\mathbb{F}$.

$$F_M = \Psi_M\left(\mathbb{F}\right) \tag{5}$$

The next rules show how the Filter Function is expanded by logical expressions on M. The formal proof of the following rules follows from the definition of filter function and due to the space limitation we simply state them.

$$\Psi_{M_1 \wedge\ M_2}\left(F\right) = \ \Psi_{M_1}\left(F\right) \cap \Psi_{M_2}\left(F\right) \tag{6}$$

$$\Psi_{M_1 \vee\ M_2}\left(F\right) = \ \Psi_{M_1}\left(F\right) \cup \Psi_{M_2}\left(F\right) \tag{7}$$

$$\Psi_{M_1} \mathrm{o}\ \Psi_{M_1}\left(\mathrm{F}\right) = \Psi_{M_1}\left(\Psi_{M_2}\left(F\right)\right) = \ \Psi_{M_1 \wedge M_2}\left(F\right) \tag{8}$$

Network Space: A region in $\mathbb{F}$ that binds all packets coming from or going to a participating network and is represented by F_N. The filter function $\Psi_M\left(\mathbb{F}\right) = F_N$ that filters all packets in F_N is called the binding function of N and M is called the binder expression of N.

Subnet: Network N_1 is a subnet of Network N_2 if and only if the space of N_1 is a subset of the space of N_2:

$$F_{N_1} \subseteq\ F_{N_2} \iff\ N_1 \ll\ N_2 \tag{9}$$

Note that "$\ll$" denotes subnet relation.

2) Control Policies and Pipeline: In HSA[12] any packet traversal through networking boxes is modeled as transformation over the flow space. We use the same notion to model control policies defined by InPs or VN owners at HyperExchange. A policy in general can be a sequence of OpenFlow actions or steering traffic through a custom VNFs. An OpenFlow header modification transforms the packet in $\mathbb{H}$ while dropping the packet or sending it out of a port is transformation in $\mathbb{P}$. Note that a packet drop is modeled by assigning a logical port to the packet. Based on this notion a control policy can be expressed as chain of transformation functions in $\mathbb{F}$:

$$P\left(F\right) = T_1\left(\ldots T_n\left(F\right)\right) = T_1 o\ \ldots o\ T_n\left(F\right) \tag{10}$$

This chain can include filter function as well to apply the policy on a specific slice of traffic. Consider the traffic coming from VN1 in InP1 and going VN2 in InP2 and consider M1.1, M1, M2.2 and M2 as the binder expression of these networks respectively; then the incoming pipeline can be modeled as follows:

$$\rho_{in} = P_{VN1}\ o\ \Psi_{M1.1}\ o\ P_{InP1}\ o\ \Psi_{M1}\left(\mathbb{F}\right) \tag{11}$$

And the final outgoing traffic will be:

$$\rho_{out} = P_{InP2}\ o\ \Psi_{M2}\ o\ P_{VN2}\ o\ \Psi_{M2.2}\left(\rho_{in}\left(F\right)\right) \tag{12}$$

3) Policy Authorization: Since a policy is modeled as a transformation in flow space, a policy authorization indicates the set of allowed transformations. It is assumed that all networks without subnet relation are isolated:

$$F_{N_1} \not\subset F_{N_2} \Rightarrow \ F_{N_1} \cap F_{N_2} = \emptyset \tag{13}$$

A transformation is allowed if it keeps the packet in the same network space.

$$P\,(F) : \begin{cases} F_N \rightarrow F_N \Rightarrow Allowed \\ otherwise \ \Rightarrow Not\ Allowed \end{cases} \tag{14}$$

Transformation along network spaces can be allowed if the principal of the policy has ownership over both networks.

This formal model helps us to present a precise and general definition of VNs. It abstracts away the internal protocols and topologies of participating networks and hence, it is the basis of a pipeline design and can even be extended to include ICNs and name-based routing. We also used the model to drive a logic for authorization. The notion of binding helps us to guarantee the isolation of policies and can be extended to include custom ABAC authorization policies.

B. Network Specification Data Model

We have defined a data model for network specification at exchange point based on the geometric model described. For simplicity, one level of network hierarchy is considered in this model. Thus, at the top level we have InP networks and at the next level VNs can be defined as subnets of InPs. The data model is JSON structured with following main values:
Network ID: A unique identifier for the network.
Network Domain: If this network is a VN, network ID of InP will be specified as the domain. Network domain will not be specified for InP networks.
Binder Expression: A match expression that filters all packets of this network at exchange point. The binder is a list of lists modeling the expression in form of "sum of product" (i.e. OR of ANDs).
Metadata: A set of key values that describe additional attributes of the network.

The representation of this data model for service provider 2 in Figure 1 is:

```
{ net_id: InP2,
      net_domain: None,
      binder:{{port:2}},
      metadata: {}
}
```

Consider VN2 in the Figure 1 as private IP network with address range of 192.168.0.0/16. Then network data structure for this network will be:

```
{ net_id: VN2,
      Net_domain: InP2,
      binder: {{ip:"192.168.0.0/16"}}
      metadata: {}
}
```

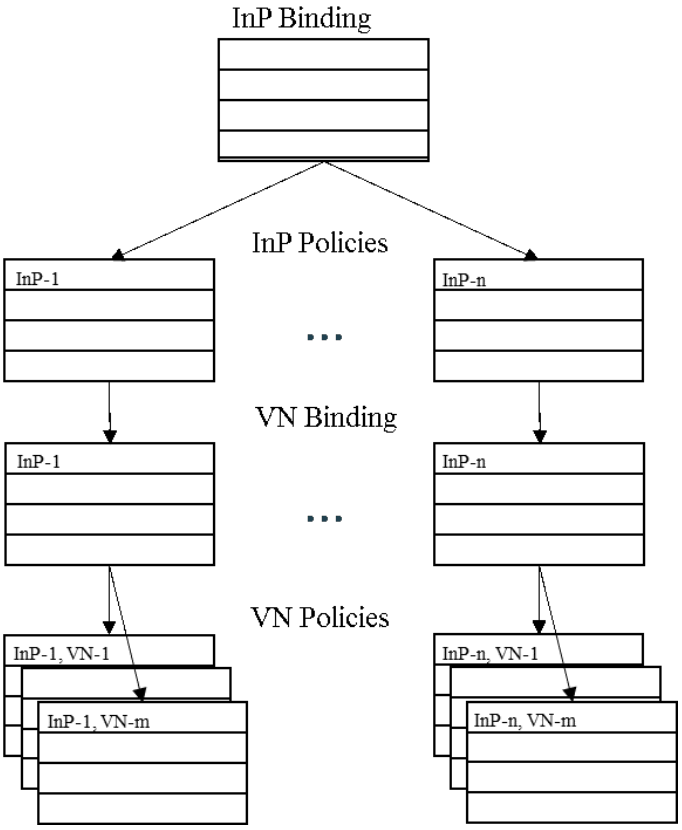

Fig. 4. Main steps of the switching pipeline

III. DATA PLANE STRUCTURE

HyperExchange is architectured with a three-layer data plane that is inspired by our SDI reference model. The bottom-most layer is the hardware switching fabric connected to a set of server racks on top. Each rack hosts a software switch (i.e. OVS) and an OpenStack[13] agent on top of that. The set of Open vSwitches forms the second layer of a data plane which is the software switching fabric. These software switches are mainly used to establish steering circuits through Virtual Network Functions (VNF) and also as secondary flow store for swapping flow tables with hardware switches. The upper-most layer is the VNF-plane which is a set of Virtual Machines (VMs) hosting standard (validated) software network functions. VMs in each agent are connected to the OVS through Virtual Ethernets (Veth pairs).

A. Traffic Switching Pipeline

A packet processing pipeline is designed to realize the concept of formal model described in the previous section. To address the exclusion of incoming and outgoing flow spaces, the pipeline is designed with two separate phases. Since the outgoing phase has technically the same steps (in reverse direction) as the incoming phase, we only describe the incoming phase and we refer to the incoming phase as the pipeline.

The pipeline should enforce policies for InPs and their tenants separately. A logical priority is considered for InPs over their hosted VNs and the control policies of the provider should

be applied on the traffic prior to applying tenant policies. To make a clear separation between packets from different participating networks (including provider network and tenant network) our design has benefited from the multi-table feature in OpenFlow 1.3. A separate flow table is used for each of the participating networks.

Figure 4 demonstrates the overall life cycle of a packet in the pipeline which has the following four main steps.

1) Traffic Binding to InP Networks: Packets coming to the exchange point will be matched by the binder of InP networks at the very first step. This is the technical realization of binder function described in previous section, using OpenFlow match. the binding process is actually done by matching all packets by a flow-table containing binder flow-entries of all InP networks. In case of match, the packet will be sent to the policy table of the matched InP. A packet in this step must be bound to at most one InP network (is const). If a packet does not match to any InP network binder, it will be dropped.

2) InP Policy Enforcement: There is a dedicated flow table per each service provider. Filters and actions defined by an InP will be stored as flow entries in its dedicated policy table. It is common that a packet does not match to any entry in the policy table of InP; that means the InP has not defined any policy. In this case, the packet will be sent directly to next table in the pipeline line. This can simply be done by defining flow entry with the least priority to match on all packets and send them to the next table as the action.

3) Traffic Binding to VNs: VN network binding is similar to InP binding. There is a separate tenant binding (i.e. a flow table for tenant network binding) dedicated to each InP. There is a flow table for tenant network binding per each service provider. After enforcement of policies defined by associated InP, packets will be matched by the VN binding table of the same InP. Once the packet is matched to a VN in the InP, it will be sent to the policy table of that VN.

4) Tenant Policy Enforcement: Similar to InP policy tables, there is a dedicated table for each tenant of each InP. These tables contain flow-entries associated with the policies defined for each VN. In contrast with InP policy tables, a packet cannot continue the pipeline without matching any entry in the VN policy table and in that case the packet will be dropped.

B. Design Challenges

Here we mention and address two important challenges of the design described above, in terms of scalability and feasibility.

1) Virtual Flow Tables: The primary approach of dedicating a flow table for each InP network as well as each VN network can cause a severe growth in the number of flow tables needed. Due to the limited number of flow tables supported in a real OpenFlow switch, this can cause an important scalability problem for our design. To address this issue we introduced a novel approach called Virtual Flow Table (VFT) with which we can store multiple flow tables in a single real flow table. Metadata in OpenFlow 1.3 is used to realize this. Each VFT is assigned with a Virtual Table ID (VTID). In the primary case once a packet matches to a network binder the packet will be sent to the table associated with that network. In this case, once the packet is matched in the binding table VTID of the designated network will be set as the Metadata and packet will be sent to the next table which stores all tables of a group (for example policy tables of all service provider networks). In this table, all flow entries of a Virtual Table have metadata = VTID as an additional match criterion. As an example, consider a table containing all of the policies of all InPs. The policies of each InP will be differentiated by an extra match criterion that is VTID of that InP. By using this approach, the total number of flow tables can be greatly decreased to a fixed number.

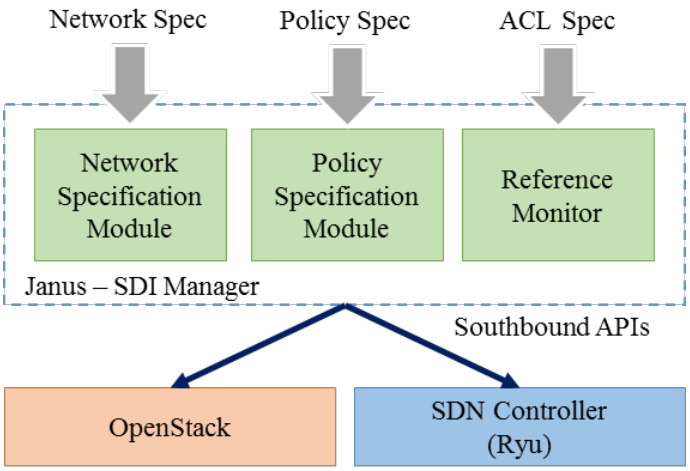

Fig. 5. Overall architecture of HyperExchange control-plane

2) Circuit Switched VNFs: HyperExchange allows users to steer traffic through Virtual Network Functions (VNF). However, the steering mechanism is a challenge in this design. OpenFlow logical ports can be used to establish a dedicated circuit to each middlebox. This will allow packets to be forwarded only based on incoming port in a service chain. A logical port is a standard type of OpenFlow ports that can be used to create logical links. The concept of logical ports in OpenFlow protocol is neutral to the technology being used to realize it. For example, VXLAN tunnels can be used to establish logical links. If a user specifies VNF steering in the control policy, the SDI manager (described in next section) will create a dedicated logical port in the main switch and stablish a tunnel between the logical port and the VM running the VNF. This design abstracts away the topology details of HyperExchange for users.

IV. CONTROL PLANE ARCHITECTURE

The control plane of HyperExchange is an extension of our reference SDI manager design[14]. Figure 5 shows the overall architecture of control plane which includes SDI-manager and the modules added specifically for HyperExchange. The SDI-manager has southbound APIs to the SDN controller (Ryu) which controls Hardware and Software OpenFlow switches. The other southbound API of SDI-manager is to the Cloud controller (OpenStack). Through this API VMs will be provisioned on demand to host requested VNFs based on user policies.

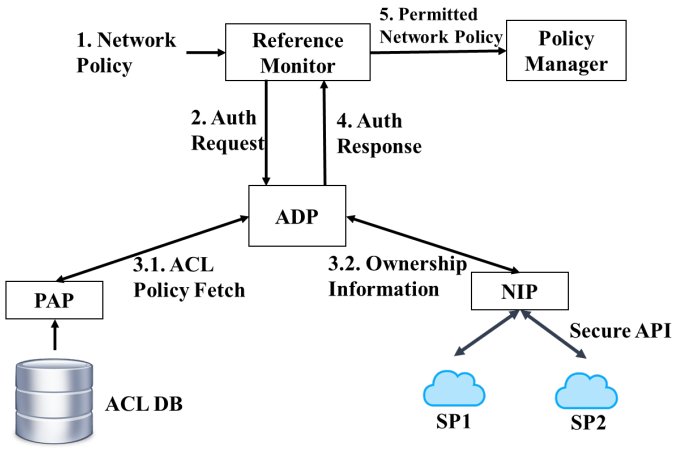

Fig. 6. Authorization process in the Reference Monitor Module

A. Main Modules

The control-plane of HyperExchange includes three main modules and each provides its own associated northbound APIs.

1) Network Specification Module: Any participating network must be specified through this module. InP networks will be defined statically and cannot be defined by user requests through API, but tenants of predefined InPs can define arbitrary VNs through the API. The API specification is the JSON data-model described in section 2. Once a network is defined, this module extracts attributes from the binder list of the specified VN and sends an authorization request to the Reference Monitor to authorize the VN. In case of successful authorization, the network specification module creates binding flow-entries from the specifies binder list and installs them to the switch.

2) Policy Specification Module: The user defined policies for network control will be received by Policy Manager (PM). Upon receiving a policy, the PM module extracts policy attributes and creates an authorization request to the Reference Monitor. If the response is "permitted", the policy will be stored in database and the policy flow table in switch.

3) Reference Monitor: This module is the reference monitor for authorizing network control policies at HyperExchange. The authorization system relies on ownership information that indicates the subset of authorized flow space of each user. Since VNs from different InPs can participate in HyperExchange, a single static root of trust cannot be used to indicate ownership information. To address this issue and provide more dynamic and extensible authorization, we have designed an Attribute Based Access Control (ABAC) system for HyperExchange. The overall architecture of this system is shown in Figure 6. This architecture is inspired by XACML[15] and includes three main points. Authorization Decision Point (ADP) receives authorization requests from other control-plane modules. It then retrieves Access Control Policies from Policy Administration Point (PAP) and network ownership information from Network Information Point (NIP). NIP can get the ownership information of tenant VNs from their InP through API on a secure channel.

B. Corner Issues

Here we mention and address two important corner issue regarding the described design for control-plane.

1) Unified Identity: The authorization system of HyperExchange requires getting ownership information of a tenant VNs from InPs. However, each of the InPs and the HyperExchange itself have a local identity management system. A single user can have an identity in each of these systems. Unifying different identities of a single real user is a fundamental challenge for authorization at HyperExchange. Our primary solution for this problem is to use a centralized identity provider for all of the participating InPs and exchange point. In this case all of the service providers and the exchange point will authenticate tenants through the central identity provider. A single Shibboleth identity server can be used as the identity provider for all parties. In the general, all of the participating InPs may not be joined in the same identity provider. In case of multiple identity providers, an identity peering mechanism is needed which is neutral to the design of HyperExchange.

2) Network Resource Deallocation: A common feature of multi-tenant environments is the high frequency of deallocation and reallocation of the resources. An example of networking resources can be a VLAN tag or a floating IP address. Since HyperExchange relies on ownership information of networking resources from InPs to authorize tenant policies at exchange point, a resource deallocation in an InP can invalidate a previously authorized tenant policy at exchange point. For example, if Alice as a tenant allocates IP1 = 142.150.208.235 in SAVI as the InP, her policies over IP1 will be authorized and enforced in flow tables of the exchange point. However, once she releases IP1 in SAVI, her previously defined policies that include IP1 are no longer valid at the exchange point. This example shows that exchange point must be aware of resource deallocation/reallocation in the participating InPs to make sure that tenants policies are always valid. An ideal solution for this challenge could be a notification of deallocation from the InPs. However, not all of the participating InPs can provide such capability. For those service providers, a timeout and revalidation mechanism can be used for all of the specified VNs.

V. PROTOTYPE AND USE-CASE EXPERIMENT

We have implemented a prototype of HyperExchange to show the feasibility of our modeling and design. The prototype is an extension of our SDI-manager reference model called Janus written in Python. In the current implementation the user can specify a VN using a network specification API. The authorization module uses a private API to get VN attributes and to authorize the specification. By this step, the policy specification API is the primary structure of OpenFlow flow-entries; however, we are working to implement a more abstracted policy expression API to add to the system. Our current implementation supports OpenFlow actions and traffic steering through a single function as the policies described by user and traffic steering through a chain of VMs is under development. The authorization module uses remote APIs for network specification requests but for policy flow entries, it uses local specifications of network domains.

We have deployed our prototype of HyperExchange between SAVI and GENI testbeds. Our prototype deployment is built on a hardware switch between ORION and Internet2 networks, the underlying interconnection of SAVI and GENI testbeds respectively. Each of the SAVI and GENI networks are connected to the exchange point via a single dedicated physical port.

A. Experiment Case: Peering of Layer2 Networks

By use of HyperExchange it is possible to setup layer-2 networks over multi-region clouds without encapsulation. A generic virtual layer-2 network is key for any further innovation in upper layers such as IP alternatives. Also it makes it possible to simply define custom and private IP networks on a Wide Area layer-2 Network. These features make inter-domain layer-2 peering a beneficial usecase of HyperExchange. Layer2 networks in SAVI are established by end-to-end path stitching on OpenFlow switches based on MAC addresses of endpoints. On the other side GENI uses VLAN tags to create a layer-2 network between arbitrary set of VMs. Each of these InPs has a different logic to realize a Layer2 network. Thus peering of two virtual Layer2 networks on both sides as single end-to-end layer-2 network is the challenge we addressed by use of HyperExchange. Our flexible model for networks at exchange point allows us to simply define and peer these networks in a uniform manner. In our test case we had two VMs in SAVI testbed connected in a Layer2 network in Toronto with the following network data structure at exchange point:

```
{ net_id: SAVI_L2,
      Net_domain: SAVI_NET,
      binder: {{mac = fa:16:3e:65:ac:52},
               {mac = fa:16:3e:5d:33:db}}
      metadata: {}
}
```

In GENI side we defined a VLAN including a VM in Chicago with the following network data structure at exchange point:

```
{ net_id: GENI_L2,
      Net_domain: GENI_NET,
      binder: {{VLAN_TAG = 7273}}
      metadata: {}
}
```

As can be seen, the network attributes of a Layer2 network in GENI is not related to the number of nodes. However, in SAVI as the number of nodes in a Layer2 network increases, the network attributes to be authorized will also increase. Figure 7 demonstrates the relation of number of nodes and the time it takes at HyperExchange to query each InP to verify network specifications. We have emulated authorization API to GENI by users own credentials. However, an speaksfor API is under development in GENI that can be used for remote authorization on behalf of the user.

Figure 8 shows the comparison of cumulative distribution of time over 10 trials for specification of the GENI side VN. The overall time is the time of processing VN specification, authorization through remote API and creation and installation of binding flow-entries to the switch. In Figure 8 the black line

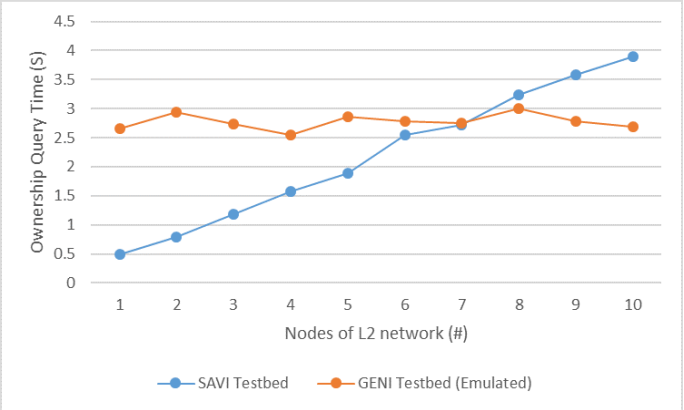

Fig. 7. Remote attribute authorization time based on the number of nodes in VN

is the time excluding remote authorization time and the dotted line is the entire time. The figure shows that a large amount of time is spent on remote authorization. Thus our technique to authorize only the network specification using remote API and authorizing later policies by local and pre-authorized domains can effectively reduce the policy installation time.

We defined the following policies to peer these VNs at HyperExchange:

```
bind(SAVI_L2).incoming()
        .modify({"type":"SET_VLAN",
        "value":7273})
        .output(GENI_L2)

bind(GENI_L2).incoming()
  .modify({"type":"STRIP_VLAN"})
  .output(SAVI_L2)
```

As mentioned earlier, policy specification must be installed as flow entries at this state of our implementation and the above code is the pseudo representation of the peering policies. As depicted in Figure 9, we have measured Round Trip Time from a VM in Chicago to a VM in Toronto for the regular path over the Internet and Layer2 directed path through exchange point. Our experiment shows that HyperExchange flexibility helps InP tenants to setup arbitrary end-to-end paths between VNs over autonomous Infrastructures.

VI. DISCUSSION AND RELATED WORK

How does HyperExchange compare with existing Software Defined Exchanges? Richter *et al.*[16] presented that how a central route server can facilitate peering at IXPs. The concept of Software Defined Exchanges (SDX) and use of SDN for flexible inter-domain networking is introduced in[2]. The primary implementation of SDX [17] enables participating autonomous systems to overwrite basic BGP route selection at IXPs. The Cardigan project [18] was another primary attempt to enhance inter-domain networking by use of SDN. Based on the measurements provided in [19] none of these primary implementations can address scalability requirement of a large scale exchange point. Industrial Scale SDX (iSDX) [19] is a more recent implementation of SDX based on Ryu controller which has addressed the scalability issues of primary

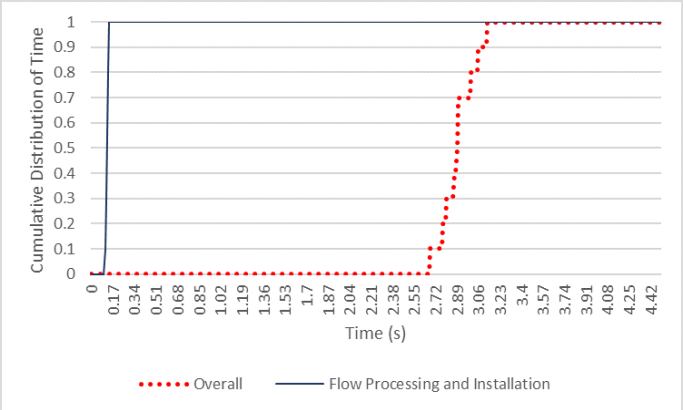

Fig. 8. Time analysis of network specification in HyperExchange (GENI Side)

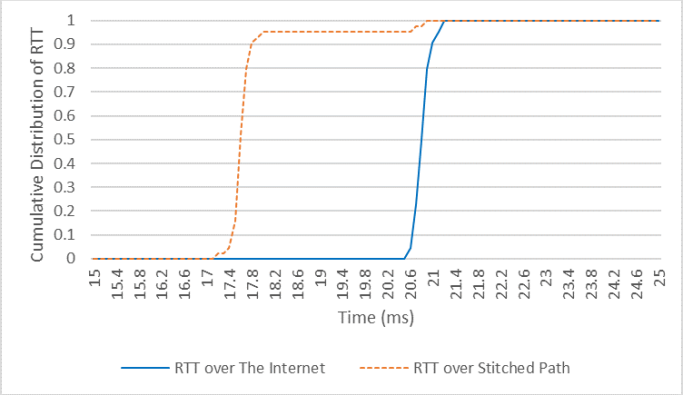

Fig. 9. RTT comparison of the regular path over the Internet vs the directed path through the exchange point

implementations by use multi-table feature of higher version of OpenFlow. All of these implementations have some features in common. Only IP networks can exchange traffic in current SDX architectures. Moreover, these implementations have not targeted exchange between multitenant environments so there no clear division between service provider networks and tenant networks.

Why is it beneficial to peer Virtual Networks? Expanding a Virtual Network over multiple Infrastructure Providers is a requirement for future services; because of the geographical distribution of the network (e.g. Akamai global CDN that is expanded over many multiple ISPs) and cost optimization strategies and federation (e.g. Apple iCloud that uses Amazon EC2 in addition to their own data-centers). In these cases, the network needs to be deployed in multiple InPs and it is required to federate these InPs and their networks. In fact the main motivation of this paper came out of a real problem that is network federation of two research testbeds (SAVI GENI). The only current solution is encapsulation over IP (i.e. overlays) which inherits the ossification of the current Internet and provides a narrow solution for specific use cases, not a holistic approach for network federation of multiple SDI domains.

How does control authorization in HyperExchange compare with other network flow authorization frameworks? Authorization of network control policies from tenants of different service provider poses new challenges in the design of exchange points. Even though the authorization of network control is discussed in FLANC[20] and used in iSDX implementation, a static root of trust is considered to have all ownership information of networking resources and it is not mentioned that how this information can dynamically be gathered from participating service providers. Particularly, in a Virtual Networking Environment, network domains of tenants may change rapidly and the Reference Monitor of the exchange point must be able to recollect these ownership information as they change in participating Infrastructure Providers. HyperExchange uses secure APIs to collect and update tenant ownership information from participating InPs.

How is HyperExchange related to Virtual Network Embeding? Virtual Network Embedding is concerned about the mapping between virtual topology and physical topology and is well studied in the literature [21][22][23]. In case where multiple InPs are involved, an end-to-end VNE platform can used to slice and map slices of VNs to each InP [24]. However, these InPs can use different logic and technologies for network virtualization. For instance, there are multiple protocols available to create a virtual Layer-2 network including GRE, VLAN, VXLAN and MPLS and each of them might be used in one of the InPs. However, the nature of a Layer-2 network is the same and the exchange point must be able to peer Layer-2 networks independent of the underlying protocol. In these cases the VNE platform can employ a HyperExchange to enable traffic exchange between dissimilar InPs.

VII. CONCLUSION AND FUTURE WORK

We presented HyperExchange which enables traffic exchange between Infrastructure Providers and their hosted Virtual Networks. We built a formal model for traffic classification at exchange point and extended it to design the traffic switching pipeline of HyperExchange. Our formalism allowed us to define a protocol-agnostic network model that satisfies the feature of protocol customizability of Virtual Networking Environemnts. Based on the formal specifications, we proposed an extensible architecture for the switching fabric of the exchange point. Our prototype is still under extension to provide multi-VM service changing. The policy API needs more extension for further expressiveness. Our current design for authorization system is inspired by XACML but does not fully support its standards.

As a major research direction for the future, we are interested to design an end-to-end orchestration platform based on HyperExchange that can manage virtual networks deployed over multiple cloud domains.

ACKNOWLEDGMENT

This work and all the published papers which follow are funded in part or completely by the Smart Applications on Virtual Infrastructure (SAVI) project, funded under the National Sciences and Engineering Research Council of Canada (NSERC) Strategic Networks grant number NETGP394424-10.

REFERENCES

[1] K. R. Butler, T. R. Farley, P. McDaniel, and J. Rexford, "A survey of BGP security issues and solutions." vol. 98, no. 1, pp. 100–122.

[2] N. Feamster, J. Rexford, S. Shenker, R. Clark, R. Hutchins, D. Levin, and J. Bailey, "Sdx: A software defined internet exchange," p. 1.

[3] B. Ager, N. Chatzis, A. Feldmann, N. Sarrar, S. Uhlig, and W. Willinger, "Anatomy of a large european IXP," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM.

[4] J.-M. Kang, H. Bannazadeh, H. Rahimi, T. Lin, M. Faraji, and A. Leon-Garcia, "Software-defined infrastructure and the future central office," in *2013 IEEE International Conference on Communications Workshops (ICC)*. IEEE, pp. 225–229.

[5] J. Kunz, C. Becker, M. Jamshidy, S. Kasera, R. Ricci, and J. Van der Merwe, "OpenEdge: A dynamic and secure open service edge network," in *IEEE/IFIP NOMS*.

[6] A. Panda, J. M. McCauley, A. Tootoonchian, J. Sherry, T. Koponen, S. Ratnasamy, and S. Shenker, "Open network interfaces for carrier networks," vol. 46, no. 1, pp. 5–11.

[7] N. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Communications magazine*, vol. 47, no. 7, pp. 20–26, 2009.

[8] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "SAVI testbed: Control and management of converged virtual ICT resources," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, pp. 664–667.

[9] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," vol. 61, pp. 5–23.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," vol. 38, no. 2, pp. 69–74.

[11] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and others, "P4: Programming protocol-independent packet processors," vol. 44, no. 3, pp. 87–95.

[12] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pp. 113–126.

[13] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," vol. 55, no. 3.

[14] B. Park, T. Lin, H. Bannazadeh, and A. Leon-Garcia, "Janus: design of a software-defined infrastructure manager and its network control architecture," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. IEEE, pp. 93–97.

[15] S. Godik, A. Anderson, B. Parducci, P. Humenn, and S. Vajjhala, "OASIS eXtensible access control 2 markup language (XACML) 3."

[16] P. Richter, G. Smaragdakis, A. Feldmann, N. Chatzis, J. Boettger, and W. Willinger, "Peering at peerings: On the role of IXP route servers," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, pp. 31–44.

[17] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: a software defined internet exchange," vol. 44, no. 4, pp. 551–562.

[18] J. P. Stringer, Q. Fu, C. Lorier, R. Nelson, and C. E. Rothenberg, "Cardigan: Deploying a distributed routing fabric," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, pp. 169–170.

[19] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, "An industrial-scale software defined internet exchange point," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pp. 1–14.

[20] A. Gupta, N. Feamster, and L. Vanbever, "Authorizing network control at software defined internet exchange points."

[21] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[22] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 783–791.

[23] M. R. Rahman, I. Aib, and R. Boutaba, "Survivable virtual network embedding," in *International Conference on Research in Networking*. Springer, 2010, pp. 40–52.

[24] M. Chowdhury, F. Samuel, and R. Boutaba, "Polyvine: policy-based virtual network embedding across multiple domains," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*. ACM, 2010, pp. 49–56.