Towards Flexible, Scalable and Autonomic Virtual Tenant Slices

Mohamed Fekih Ahmed, Chamssedine Talhi, and Mohamed Cheriet
École de Technologie Supérieure (ÉTS), Montreal, QC, Canada
mohamed.fekih.ahmed@synchromedia.ca, {chamsseddine.talhi, mohamed.cheriet}@etsmtl.ca

***Abstract*—Multi-tenant flexible, scalable and autonomic virtual networks isolation has long been a goal of the network research and industrial community. For today's Software-Defined Networking (SDN) platforms, providing cloud tenants requirements for scalability, elasticity, and transparency is far from straightforward. SDN programmers typically enforce strict and inflexible traffic isolation resorting to low-level encapsulations mechanisms which help and facilitate network programmer reasoning about their complex slices behavior.**

In this paper, we propose SD-NMS, a novel software-defined architecture overcoming SDN and encapsulation techniques limitations. SD-NMS lifts several network virtualization roadblocks by combining these two separate approaches into an unified design. SD-NMS design leverages the benefits of SDN to provide Layer 2 (L2) isolation coupled with network overlay protocols with simple and flexible virtual tenant slices abstractions. This yields a network virtualization architecture that is both flexible, scalable and secure on one side, and self-manageable on the other. The experiment results showed that the proposed design offers negligible overhead and guarantees the network performance while achieving the desired isolation goals.

Keywords: Isolation, Flexibility, Scalability, Self-Management

I. Introduction

Along with the spread of Cloud Computing, network virtualization is highly used in datacenter networks (DCNs). The popularity of virtualization results from flexible and efficient management, it makes the infrastructure of network providers more profitable by sharing expensive network devices (e.g., routers, switches) between multi-tenants network. In DCNs, the first major challenge is to isolate virtual networks of large number of tenants.

In addition to adopting network virtualization, the dynamic and elastic nature of cloud computing is challenging many aspects of network virtualization including isolation tools and technologies which are no longer practical, efficient, or flexible enough for today's tenant requirements (e.g., low level encapsulations: VLANs/GRE, Hypervisor, Middleboxes: FWs/IDS/IPS). These traditional isolation mechanisms are no more suitable for the cloud elastic environment. Network architectures limitations are attributed to big-data workload setting, traffic changing and resources sharing among between virtual tenant networks. They suffer of the following drawbacks: (i) scaling the network to the sizes needed by service providers is very expensive, (ii) limited support for multi-tenancy as they do not give the possibility to the tenant for designing his virtual networks and defining its own Layer 2 and 3 spaces, and (iii) complex and manual management operations of large set of network nodes including switches, routers and firewalls.

Novel approaches for network virtualization which are not based on traditional mechanisms have become a real possibility since the appearance of Software-Defined Networking (SDN). Providing multi-tenancy isolation by creating virtual tenant slices (e.g, FlowVisor [1], HyperFlow [2], Onix [3]) is a very challenging issue that has not been completely solved before the introduction of SDN mechanisms like OpenFlow (OF) [4]. SDN have gained a lot of attention due to their flexibility for creating separate and independent virtual networks on top of physical network infrastructures.

In this paper, we present SD-NMS, a novel software-defined architecture enabling multi-tenant scalable, flexible and autonomic isolation for virtual networks. SD-NMS provides L2 isolation with simple and flexible virtual tenants slices (VTSs) abstractions in an automatic and dynamic way. To overcome SDN scalability bottleneck and overlays protocols limitation to a single slice (See Section II), SD-NMS lifts several roadblocks by combining these two separate technical approaches into an unified design. This yields a network virtualization architecture that is both flexible, and secure on one side, and scalable on the other. SD-NMS exploits the high flexibility of software-defined components and the scalability of new overlay protocols (e.g., Virtual eXtensible LAN (VXLAN)) to create several thousands of VTSs on top of shared network infrastructures. For self-manageable VTSs requirement, SD-NMS is based on autonomic communication [5] which can be a complementary approach to SDN to evolve the neglected management plane and provide self-aware network configuration. It requires only a small amount of lines as extended application to OF controller. Using hierarchical and distributed OF controllers, we succeed to enforce our flexible isolation model and provide in the same manner an efficient and scalable offloading of control functions without losing the SDN centralized advantage. By delegating VTSs frequent and local packets to tenant controller, we limit the overhead on centralized controller that processes only global and rare events to maintain network-wide view.

The rest of this paper is organized as follows. Section II investigates issues and lacks of previous solutions addressing multi-tenancy isolation. In section III, we describe the flexible L2 isolation model proposed for SD-NMS and detail how it can be exploited to provide VTSs scalability. Section IV, we describe the SD-NMS design and detailing the composition of SD-NMS planes including the autonomic management plane. In section V, we represent SD-NMS evaluation results. Finally, section VI draws conclusions

II. Related Works

A. Research Background

Some approaches concerning the concept of multi-tenancy isolation have been proposed for network virtualization before the introduction of SDN. For example, Cabuk et al. [6] presented prototype of automated security policy enforcement for multi-tenancy based on the concept of Trusted Virtual Domains (TVDs). Their approach allows to group VMs belonging to a specific tenant dispersed across multiple Xen Hypervisor into a TVD zone. Tenant's requirements for isolation are automatically enforced by a privileged domain (DOM-0). Such solution offers tenants separation using VLAN, EtherIp and VPN tagging. Their solution presents a significant step towards: (i) tenant transparency by automating the deployment and mapping of tenants desired network topology, and (ii) isolation elasticity by orchestrating TVDs through a management framework that automatically enforces isolation necessary changes (e.g., load balancing, migration) among different hosts' hypervisors.

Based on network virtualization paradigms, SDN comes with better alternative than using the TVD privileged domain, which is not recommended against attacks. By decoupling control and data planes, SDN offers a new way to create transparent and isolated virtual networks by dividing, or slicing, the network resources. Several approaches were proposed by the SDN community. FlowVisor [1] is a project working on developing virtual network slicing in hardware programmable router. The goal of FlowVisor was the implementation of an OF proxy to allow multiple researchers to share the same network resources. Such mechanism aims to separate researchers' slices called "Flowspaces" and lets each slice to be managed by a single and independent controller. Nevertheless, these slices are completely independent and FlowVisor does not consider the tenant's virtual networks scalability and collaboration. HyperFlow [2] has a complementary approach. It introduces the idea of enabling the interconnection between separated slices. HyperFlow uses multiple controllers to manage each tenant slices following the same concept as FlowVisor. The connection between slices is provided by a shared publish/subscribe system because controllers use to update the network state and send commands to the other controllers. This mechanism does not support routing over slices and either the slices scalability. Similar to HyperFlow, Onix [3], is a distributed control platform that facilitates implementation of distributed control planes. It provides control applications with a set of general APIs to facilitate access to network state, which is distributed over Onix instances.

Previous SDN researchers have successfully involved tenant in network control which has been a long goal of the infrastructure providers. However, they have made use of centralized or distributed controllers to achieve strict and inflexible isolation between different tenant's slices and resorting to low-level encapsulations mechanisms (e.g., VLANs, GRE) which help and facilitate network programmer reasoning about their complex slices behavior. Unfortunately, there is no mechanism to satisfy tenant requirements of today (e.g., slice scalability, inter/intra slices communication and collaboration, load-balancing, migration, etc ...). In addition, there have always concerns about the SDN scalability bottleneck which has major drawbacks including lack of either network or controller scalability or both. As the network scales up, both the number of physical and virtual switches increases to guarantee the minimal QoS to the end host. Regardless of the controller capacity, a controller like NOX [7] does not scale as the network grows. Specially, SDN community [8] estimates that large DCNs consisting of 2 million VMs may generate 20 million flows per second and current OF controllers can handle 10^3 flows per second. The SDN controller becomes a key bottleneck for the network scalability.

B. *Industrial Background*

For industrial implementation, network virtualization approaches are ideally based to provide the transport by the physical network and the virtual machine service by the hypervisors. The traditional slicing technique has employed VLAN to isolate tenants' machines on a single L2 virtual network. However, this simple isolation approach depends heavily on routing and forwarding protocols and is not easily configured. VLAN management complexity imposes limitations on cloud nature and services. More importantly, VLAN lacks scalability resulting to a segmentation capacity limit to 4K tenants.

Table I summarizes the competing multiple Overlay Transport Virtualization (OVT) as alternative technologies to substitute VLAN. These technologies have been proposed within industrials that are in contrast with the open standards used in OF solutions. They use Open vSwitch (OVS) plus typical (L2/L3) physical switch to provide virtual networking isolation and tunneling unlike VLAN which ignores and dumbs OVS (e.g., VMware's vCloud Director Networking Infrastructure (vCDNI) [9], HP's NVGRE [10], Nicira's Network Virtual Platform (NVP) [11]). More recently, Cisco's Virtual eXtensible LAN (VxLAN) [12] has been adopted within several network vendor for scalable LAN segmentation and automated provisioning of logical networks between data centers across L3 networks.

The major inconvenience of overlay technologies (e.g., VxLAN, NVGRE, vCDNI) is the missing of control plane. They can support only one Overlay Virtualized Networks (OVN) due to the lack of

TABLE I: *Comparison of industrial network virtualization solutions*

OVT	Encapsulation	Control Plane	VLAN limit	Bridging	VM MAC visible	vNet Flooding	Multicast Flooding	Network State
VxLAN	L2-over-UDP	✗	✗	✗	✗	✓	Some	IP Multicast
NVGRE	MAC-over-IP	✗	✗	✗	✗	✓	✓	✓
vCDNI	MAC-in-MAC	✗	✗	✓	✗	✓	✓	Hypervisors' MAC
Nicira NVP	MAC-over-IP	✓	✗	✗	✗	Some	✗	✗

scalability. They are competing encapsulations with minor technological difference (e.g., TCP offload, load balancing, Security features) and no one supported by legacy systems. The key mechanism of scalable architecture is the control plane which maps remote VMs MAC address into a transport network IP address. These technologies face a crucial problem to determine the VM destination's IP address.

Nicira's NVP seems to be solving the problem with better scalable solution than VxLAN using an OF controller to install MAC-to-tunnel forwarding rules on OVS. Nicira's NVP extends the SDN/OF standard and introduces OVS protocol to configure OF switches. Through user-space OVS database (OVSDB), it keeps track of the topology existing tunnels. This approach scales better than VxLAN but causes network overall performance degradation.

With SD-NMS, we believe that we can lift several roadblocks by combining SDN network virtualization approaches and OVN techniques. More precisely, SDN addresses flexibility, isolation, and manageability through network slices, but suffers from scalability limitations. Network overlay techniques overcome scalability issues of cloud isolation mechanisms, but remain limited to single slice. In addition, both solutions do not offer a complete transparent isolation and lack the ability to use multiple network slices within individual SDN programs. We can overcome their respective limitations into a unified design and arrive to a consistent solution that consider both networks and OF control plane scalability bottleneck. This combination enables to reap the benefits of SDN slices while preserving scalability. This yields a network virtualization architecture that is both flexible, secure, and manageable on one side, and scalable on the other.

III. SD-NMS's Virtual Tenant Slices Abstraction Model

In this section, we present our SD-NMS's Virtual Tenant Slices (VTSs) abstraction model in details. We make our approach suitable for the cloud elastic environment and applicable for general tenant requirements. It enables tenant to: (i) easily update and scale (in/out) his allocated VTSs by adding new virtual network resources, (ii) share allocated switches ports, tables, and links between his VTSs, and (iii) easily transport his VMs to new location while maintaining the isolation proprieties.

A. *SD-NMS's L2 ISOLATION MODEL*

SDN slicing technique typically enforces strict and inflexible traffic isolation using a hypervisor sitting between SDN planes as extension to OF switches. SDN slices enforce basic isolation properties by translating, inspecting, rewriting, and policing OF entries received from tenant controllers (e.g., no flows originating from tenant$_1$ slice can reach tenant$_i$'s slices). This isolation layer added between SDN planes provide a means for infrastructure provider to limit the scope of tenant, restricting the network resources that may be affected by the tenant configuration.

We take a step towards providing rich and extensible SDN slices by presenting high level L2 isolation abstraction which divides the shared network into VTSs. We lift SDN and overlay technologies limitations by combining them into an unified design. With such combination, we enable to reap the benefits of SDN slices while preserving scalability. The results are recursive, solving overlay approach by adding the missing control plane, and overcoming both virtual networks and SDN control plane scalability bottleneck by coupling SDN slices with overlay protocols.

Definition 1 (Datapath). *Each physical or virtual OpenFlow switch, in SDN data plane, represents one datapath (dp). The set of all datapaths (dp) is denoted as* $\mathcal{D}$=$\{dp_0, dp_1, \ldots, dp_u\}$; $u = |\mathcal{D}| \geq 1$. *Each* dp_i *is composed of set of OpenFlow ports, tables and group tables, denoted respectively* $\mathcal{P}_+^{(i)}$, $\mathcal{T}_+^{(i)}$,*and* $\mathcal{GT}_+^{(i)}$.

$$dp_i = \{\mathcal{P}_+^{(i)},\ \mathcal{T}_+^{(i)},\ \mathcal{GT}_+^{(i)}\}\ /\ \mathcal{P}_+^{(i)} \subset \mathcal{P},\ \mathcal{T}_+^{(i)} \subset \mathcal{T},\ \textit{and}\ \mathcal{GT}_+^{(i)} \subset \mathcal{GT},\ \textit{where:}$$

- $\mathcal{P}$=$\{p_0, p_1, \ldots, p_k\}$; $k = |\mathcal{P}| \geq 1$: the set of all OpenFlow switches' ports in the multi-tenant data center. These ports can be classified into two types. The first class ($\mathcal{P}_{ext}$) is composed of the external ports which link between switches ($p_{i;i=1..k} \in \mathcal{P}_{ext}$). It must be shared between multi-tenant network to forward packets to external networks or neighbours. The second class ($\mathcal{P}_{in}$) regroups the edge or internal port which links OpenFlow port to virtual machine's virtual network interface (VNI) ($p_{i;i=1..k} \in \mathcal{P}_{in}$). Therefore, $\mathcal{P}$ can be defined as $\mathcal{P} = \mathcal{P}_{in} \cup \mathcal{P}_{ext}$.
  A port of an OpenFlow switch is considered as boolean vector: $p_i = (status,\ type,\ dedicated) \in \{0,\ 1\}^3$. The first class $status$ indicates the activation status of port ($p_i(0) = 0$ for inactivated port and $p_i(0) = 1$ for activated one). The arriving packets to this port will be automatically dropped. For the $type$ value, If $p_i(1) = 0$ then $p_i \in \mathcal{P}_{in}$, otherwise $p_i \in \mathcal{P}_{ext}$. Finally, the $dedicated$ class indicates if the port is dedicated only for one tenant ($p_i(2) = 1$) or shared between multi-tenant network ($p_i(2) = 0$).
- $\mathcal{T}$=$\{t_0, t_1, \ldots, t_h\}$; $h = |\mathcal{T}| \geq 1$: the set of OpenFlow tables where:
  $t_{i;i=1..h}$ is boolean vector: $t_i = (status,\ dedicated) \in \{0,\ 1\}^2$. $t_i(0) = 0$ points out inactivated table and $t_i(0) = 1$ for activation. The $dedicated$ class indicates if the table is dedicated only for one slice ($t_i(1) = 1$) or shared between multiple tenant slices ($t_i(1) = 0$).
- $\mathcal{GT}$=$\{gt_1, gt_2, \ldots, gt_l\}$; $l = |\mathcal{GT}| \geq 1$: the set of OpenFlow group tables [1]. Each $gt_{i;i=1..l}$ is handling similar or common actions. Similar to OpenFlow table, gt_i is boolean vector: $gt_i = (status,\ dedicated) \in \{0,\ 1\}^2$.

We define a set of controllers $\mathcal{C}$=$\{c_1, c_2, \ldots, c_r\}$; $r = |\mathcal{C}| \geq 1$ that can be connected to one or more OpenFlow switches. Each controller, $c_{i;i=1..r}$ can manage one or more VTS.

Definition 2 (Multi-tenant Network). *Multi-tenant Network is a set of virtual networks dedicated for multi-tenant, denoted as* $\mathcal{VTM} = \{VTN_1, VTN_2, \ldots, VTN_w\}$; $w = |\boldsymbol{VTM}| \geq 1$.

Definition 3 (Virtual Tenant Network). *Virtual Tenant Network is a set of virtual tenant slices, denoted as* $\mathcal{VTN}$=$\{VTS_1, VTS_2, \ldots, VTS_q\}$; $q = |\mathcal{VTN}| \geq 1$. *We allocate for each* $\mathcal{VTN}$ *a dedicated and shared resources from* $\mathcal{D}$.

Definition 4 (Virtual Tenant Slice).

[1]A group table consists of group entries. The ability for a flow entry to point to a group enables OpenFlow protocol to present additional methods of forwarding (e.g. select and all). It can be used for grouping common actions of different flows or handling specific forwarding like load-balancing.

A tenant's virtual network, $\mathcal{VTN}_{i;i=1..w}$, *can be composed of one or more Virtual Tenant Slice (*VTS*) where given SDN resources allocated for tenant topology are dedicated for each slice including:*

- *a subset of OpenFlow switches ports (*$\mathcal{P}$*), denoted as :* $\mathcal{P}_-^{(i)}$ / $\mathcal{P}_-^{(i)} \subset \mathcal{P}$,
- *a subset of processing tables (*$\mathcal{T}$*), denoted as :* $\mathcal{T}_-^{(i)}$ / $\mathcal{T}_-^{(i)} \subset \mathcal{T}$,
- *a subset of groups tables (*$\mathcal{GT}$*), denoted as :* $\mathcal{GT}_-^{(i)}$ / $\mathcal{GT}_-^{(i)} \subset \mathcal{GT}$.
- *dedicated OpenFlow controllers, denoted as :* $\mathcal{C}_-^{(i)}$ / $\mathcal{C}_-^{(i)} \subset \mathcal{C}$,
- *and finally the list of the installed OpenFlow entries (*$\mathcal{FE}_-^{(i)} \subset \mathcal{FE}$*) in the allocated processing space* $\mathcal{T}_-^{(i)}$ *and* $\mathcal{GT}_-^{(i)}$ *where:* $\mathcal{FE}$=$\{fe_1, fe_2, \ldots, fe_v\}$; $v = |\mathcal{FE}| \geq 0$. $fe_{i;i=1..v}$ *can be installed on OpenFlow table* t_i *or group table* gt_i *and is dedicated for specific port* p_i. *By default, we deny all flow entering the tenant slice's port similar to basic firewall configuration.* fe_i *groups the list of all permitted flows, denoted as* $\mathcal{PF}$=$\{pf_1, pf_2, \ldots, pf_z\}$; $z = |\mathcal{PF}| \geq 0$. *Each* $pf_{i;i=1..z}$ *is detailing the permitted packet header attributes (e.g., IP source and destination, Port source and destination, VXLAN tag . . .).*

From these definitions, our L2 isolation model simply states that no network element ($p_{i;i=1..k}$, $t_{j;j=1..h}$ or $gt_{o;o=1..l}$) can be simultaneously dedicated and shared. A tenant slice could allocate entire datapath' ports and processing spaces. Furthermore, it may consist of a mix of physical and virtual ports, and multiple processing spaces (t_j and gt_o) on different and distributed OpenFlow switches. The mapping process is required to ensure VTS isolation and compatibility on underlay network. Every logical identifier must map to unique real one. This condition should be satisfied for all OpenFlow switches and their related ports, tables and groups tables. Moreover, $\mathcal{FE}$ and their respective $\mathcal{PF}$ specify the list of permitted flows entering the VTS's OpenFlow port linked to tenant's VMs. Finally, VTS can be programmed independently by tenant allocated controllers ($\mathcal{C}_-^{(i)}$) while ensuring that VTS functions are compatible with pushed $\mathcal{FE}_-^{(i)}$, and that packets do not get interfering or sticking during processing, except possibly by processing incoming flows on shared $\mathcal{P}_-^{(i)}$ which are checked on shared $\mathcal{T}_-^{(i)}$ and $\mathcal{GT}_-^{(i)}$. Even this form of interference is automatically ruled out through service providers' neutral controllers. These controllers manage the shared network resources by detecting and eliminating any possible conflicts between tenants' $\mathcal{FE}_-^{(i)}$ before pushing it down to the underlay network. This last feature is already covered by previous researches (e.g., [13], [14]).

B. HOW SD-NMS's ISOLATION MODEL MEETS THE GOALS ?

Cloud multi-tenants will have different, multiple and arbitrary virtual network services to deploy. A tenant may need to scale his VTN to an arbitrary size or modify their boundaries without getting forced into a complex and static reconfigurations. For example, new VM may be needed to join tenant's slice for arbitrary reasons such as scaling tenant applications capacity or for load-balancing purpose. For that, we facilitate such operation by simply adding new p, t and gt to the VTS allocated resources without getting forced to rebuilding steps. Also, a tenant wishing to share a VM or processing space between two VTSs might simply need to set a port, table or group table as shared network elements between them. A tenant might also want to move some VMs or entire VTS's VMs from one slice to another. Our flexible isolation supports also this need by enabling tenant to replicate the same configuration of moving VMs on the joined VTS. We simply link migrated VMs to the new OpenFlow ports added on the target VTS and then copy and adapt all their related flow entries.

Most existing SDN isolation solutions cannot meet all these needs simultaneously. SD-NMS's L2 isolation model gives each tenant a straightforward abstraction view of his allocated VTN topology and related VTSs: each VM is linked to a single and unique p_i, and one dedicated processing space (t_j and gt_o) is used to forward all flows entering and leaving the slice. Our flexible and self-manageable model (See section IV for more details) handles all tedious details related to enforcing the isolation, freeing cloud tenant from having to reason about tricky issues and all complex management operations. This model allows a tenant to design his VTSs as if it is the sole occupant of the shared infrastructure. That is, a tenant can be able to define his own slices while he can also automatically: (i) scale them to desired size by adding new instances, (ii) join migrated VMs from one to another, and (iii) apply optimizations that make efficient use of the allocated network resources by sharing them between allocated slices. The first two features are straightforward and automatically adapted by our high-level VTS abstraction. However, the last one needs to be more developed and discussed considering that several questions related to security enforcement and resources optimization can be raised. If tenant wants to share a virtual application between two VTSs (VTS_X and VTS_Y), all what he needs is to change the "dedicated" field of VM's linked p_i from "true" to "false" in the first VTS_X and automatically add this OpenFlow port to the second VTS_Y with a shared state. The questions that can be raised after these actions in order to accomplish the tenant requirement are:

- Which table (t_j) and group table (gt_o) will handle the processing of entering and leaving packets from/to this shared port (p_i)?
  1) It will be the dedicated processing space for VTS_X or VTS_Y or both by simply adding the shared application attributes (MAC/IP Addresses and TCP port) into the corresponding PF?
  2) Or, it will be redirected to a shared t_j or gt_o grouping the same FE and PF?
- Are both VTSs controlled by the same tenant controller (c)? IF not, the tenant must add the VTS_X' controller to VTS_Y in aim to handle all the shared p_i's traffic.

Considering any possible tenant choice from the above solutions, we combine all possibilities of sharing VTS's resources that can have place, not only to predict any tenant needs but also to have a rich and optimal isolation solution. Tenant's VTSs can scale to huge sizes with larger number of virtual applications or ports and accordingly the number of allocated tables, group tables and controllers. To fully realize the benefits of network resources sharing, we consider all multiplexing choices that can be used to achieve the best resource efficiency and cost savings. Increasing the scale alone, however, cannot fully minimize the total cost, on an attractive "*pay-as-you-go*" model for cloud computing. Therefore, we propose different VTS types denoted by $St_{i\in\{0..3\}}$ in Table II. Each St considers that one or more network resources types (P, T, GT, C) will be dedicated only for the deployed VTS. Thus, this constraint eliminates the possibility of sharing these dedicated resources with other VTSs.

TABLE II: SD-NMS's Virtual Tenant Slice Types

VTS Type \ Dedicated Resource	***P***	***T***	***GT***	***C***
St_0	✓	✓	✓	✓
St_1	✗	✓	✓	✓
St_2	✗	✓	✗	✓
St_3	✗	✗	✗	✗

The core idea of these types is to provide different security levels that ensure the desired isolation proprieties between tenant's VTSs while managing shared resources under mutual agreement enforcing unified security policies. Each St will enforce the tenant sharing constraints but it will have advantages and drawbacks at the same time. The following table III compares between VTS's types basing on the following criteria:

TABLE III: VTS Types Advantages

VTS Type \ Advantages	***Scalability***	***Optimization***	***Overhead***	***CIA***
St_0	✓	local	highest	high
St_1	✓	partial	$> St_2$	medium
St_2	✓	medium	$> St_3$	low
St_3	✓	overall	negligible	very low

- **Scalability**: will be always guaranteed for all St. Our isolation model supports VTS scalability. It allows adding more network resources to VTS.
- **Optimization**: A restriction mechanism is enabled by preventing the share of one or more resource types. St_0 restricts sharing all allocated resources, and thus network and compute resources optimization can be done just locally in VTS. In St_1, it is permitted to share only P, so tenant will resort to replicate the same resources configurations (T, GT and C) on VTSs sharing these P. This kind of optimization is indicated as "partial" because it can minimize the total cost. The difference between St_2 and St_1 is that sharing OpenFlow tables and controllers is permitted between VTS. Therefore, this type can offer better optimization than the previous one. Finally, the last type St_3 allows the most efficient usage of tenant's allocated resources and overall optimization suitable for the cloud "*pay-as-you-go*" model. This isolation type provides the required optimization, smoothness and performance for cloud application while ensuring the security between tenant's VTSs. The tenant can adapt the trust level and share any allocated network resources between his VTSs for any reasons and any desired combination.
- **Overhead**: Obviously, any extended mechanism to flow processing will add more overhead. Thus, enforcing each kind of St will require more additional time to check and verify restrictions on VTS. Logically, St_3 do not add any restriction on VTS definition. All allocated resources may be shared. Therefore, it will generate negligible overhead. We can conclude that more overhead will be added as much we add restrictions: $St0$ overhead $> St1 > St2 > St3$, as shown in the evaluation section V-B.
- **Confidentiality, Integrity and Availability (CIA)**: Tenant must be aware about the risks of each St on CIA before choosing. If a shared resource will be compromised, it can affect the rest of tenant network. Thus, different security levels are attributed to VTS types, from the highest (attributed to St_0) to the lowest one (attributed to St_3).

IV. SD-NMS's DESIGN

A. *SD-NMS Planes and Components*

The proposed SD-NMS architecture is depicted in Figure 1. The design is composed mainly of the following four planes:

- **Data Plane :** It includes physical servers hosting all virtualized components (OF virtual switches and tenants' virtual machines), physical routers, and other network elements.
- **Virtual Data Plane & SD-NMS VTSs :** It consists of all virtual applications hosting in servers. It represents virtual forwarding layer resources such as OF switches and SD-NMS VTSs.
- **Control Plane :** our isolation approach gives to network virtualization a specific description of abstraction to the control layer.

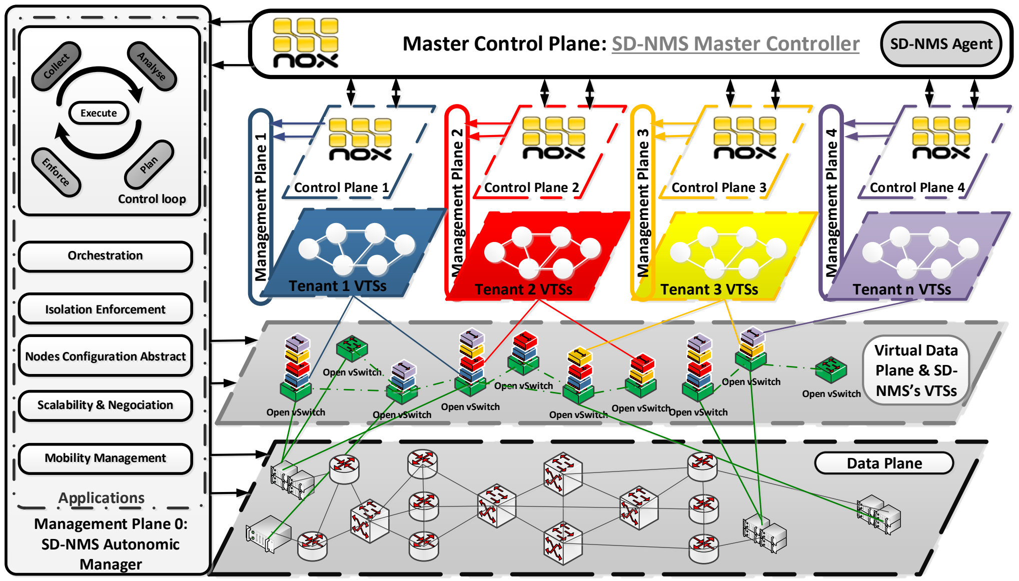


Fig. 1: *Autonomic SD-NMS Architecture Planes*

Tenant's controller is responsible for specific VTS behavior determination such as path creation, data forwarding roles, traffic engineering, etc. The Master controller is the network brain. It is responsible for global network functionalities and requirements such as multi-tenancy isolation, cross-domain communication. SD-NMS Agent (SA) is an OF application extending the OF Controller. It belongs to control plane layer and it is responsible for providing the L2 isolation by verifying incoming packets on datapath.

- **Management Plane :** there are two types of management plane in our design:
  - SD-NMS Autonomic Manager (SAM): enables an advanced and enriched self-manageability of the SDN network which is realized through number of control loops into the control plane. It is a planning, analyzing, orchestrating and provisioning entity. The analyzing entity would analyze the context and produce the required sets of control policies. Then, SAM will plan and execute the necessary management steps. The control loops are typically policy based. Once a certain condition is satisfied on the tenant's modifications and management requirements, an action will be running. Working with the Master Controller, SAM could update its global knowledge of the network and implement several functionalities such as self-provisioning, self-configuration, self-organization and self-optimization.
  - Tenant Management Planes (Management $Plane_{1toN}$): These planes give to the tenant full access to control and manage their VTSs. The desired tenant's configuration is uploaded on OF switches through the allocated controller.

B. SD-NMS Agent (SA) Design

We designed SD-NMS Agent (SA) as OpenFLow 1.3 application for SDN switches (See Fig. 2) based on extensible packet matching and pipeline processing. According to the specifications of this OF version, OF pipeline is divided into multiple sub-pipelines ($\mathcal{T}_+^{(i)}$ and $\mathcal{GT}_+^{(i)}$). The pipeline processing always starts at the first flow table: the incoming packet is first matched against flow entries of table 0. For that, we designed SD-NMS Agent to occupy the flow table 0 (SD-NMS Master Table) on each OF switch in DCN. We took this

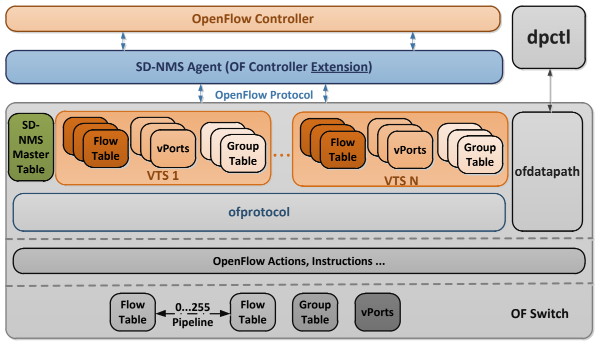


Fig. 2: *SD-NMS L2 Isolation Concept*

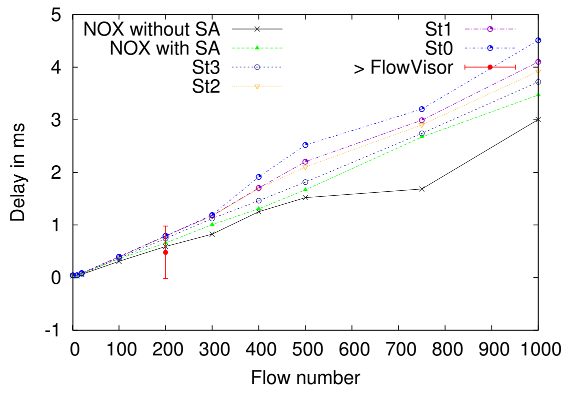
design decision based on the VTS definition which is translated into flow entries in Master Table. The rest of flows tables will be allocated or shared between SD-NMS slices. All VTSs' ports are mapped to a separated lookup table. Also, pipelines can be only dedicated or shared depending on tenant's VTS type choice. Thus, the Master Table is used as a de-multiplexer which dispatches flows to different and distributed VTSs and the extensible packet matching and the pipeline features are used by SA to provide our flexible L2 isolation model.

V. SD-NMS Evaluation

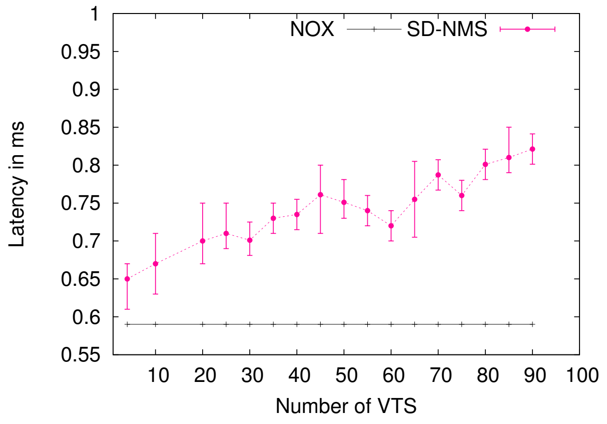
In this section, we present the deployment model of SD-NMS architecture, testbed and evaluation results.

A. SD-NMS Testbed scenario: 3-Tiers Application

We used KVM and five servers on Ericsson Blade System (EBS) interconnected with a 10 Gbps Ethernet switch to create 12 virtual machines ($VM_1 ... VM_{12}$) and 5 Open vSwitches ($S_1 \ldots S_5$). All switches supports OF 1.3 version. We run a NOX OF 1.3 controller [15] with our SA as extension which links the OF switches and



(a) Delay imposed over UDP flows by SD-NMS with different VTS Types



(b) Latency vs. VTS count

Fig. 3: SD-NMS Evaluation Results

presents the centralized SD-NMS Master controller. For each VTS, we use one NOX to control allocated resources. The switches are based on the Ericsson software implementation [16], with a modification in the forwarding plane to support OF 1.3. The NOX is based on Nicira's NOX controller, in which OpenFlow processing model is replaced with "Oflib" from the OpenFlow 1.3 Software Switch.

We deployed a simple scenario similar to the architecture in Figure 1, with one tenant and four VTSs; each one has three VMs. Tables III, IV and V represent the isolation and sharing rules in this scenario.

TABLE IV: *SD-NMS VTSs dedicated resources*

VTS_{ID}	**Dedicated resources**
VTS_1	$S_{1.t_1}$, $S_{1.t_1}$, $S_{1.p_3}$, $S_{1.p_4}$
VTS_2	$S_{1.t_2}$, $S_{1.gt_2}$, $S_{1.p_6}$, $S_{1.p_7}$
VTS_3	$S_{5.t_1}$, $S_{5.gt_1}$, $S_{5.p_2}$, $S_{5.p_3}$, $S_{5.p_4}$
VTS_4	$S_{5.t_2}$, $S_{5.gt_2}$, $S_{5.p_6}$, $S_{5.p_7}$

Notations :
$S_{j.p_i}$: $Switch_j$'s p_i, $S_{j.t_i}$: $Switch_j$'s t_i, $S_{j.gt_i}$: $Switch_j$'s gt_i

TABLE V: *SD-NMS slices shared resources*

$VTSs_{ID}$	**Shared resources**
VTS_1 & VTS_2	$S_{1.t_3}$, $S_{1.t_3}$, $S_{1.p_1}$, $S_{1.p_2}$
VTS_2 & VTS_3	$S_{5.t_3}$, $S_{5.gt_3}$, $S_{1.p_1}$, $S_{5.p_1}$, $S_{1.p_5}$
VTS_3 & VTS_4	$S_{5.t_4}$, $S_{5.gt_4}$, $S_{5.p_1}$, $S_{5.p_5}$

TABLE VI: *SD-NMS Master Controller Tables*

	OVSs Tables
SD-NMS Master Tables	$S_{1.t_0}$, $S_{2.t_0}$, $S_{3.t_0}$, $S_{4.t_0}$ and $S_{5.t_0}$

B. Evaluation

In this subsection, we evaluate the overhead generated by involving SA extended to NOX for enforcing our L2 isolation. The first experiment uses a dedicated packet generator hping to evaluate the VTS types St_i. In Figure 3a , we generate 100 UDP flows per second, then we change the rate from 1 to 1000 to compare the delay of different slicing types. The figure shows that the variation of delay comparing to NOX is negligible. Even for VTS types, there is no remarkable difference in delay between VTS types.

While it is difficult to establish direct comparison with others SDN slicing approaches like FlowVisor since they are implemented their solutions using local hypervisor on physical OF switches. FlowVisor results [1] show that including the additional isolation layer in physical switches causes an average overhead for responses of 0.48 milliseconds with 200 flow per seconds. With the same number of requests, SD-NMS has higher delay of 0.17 milliseconds (See Fig. 3a). However, this delay is acceptable seen we are using distant controller handling the isolation tasks.

In order to evaluate our approach in term of number of VTS capacity, system scalability, and performance for a larger network, we increase the number of VTSs and generate random the same rate of UDP requests from random VMs. Figure 3b shows that the latency remain reasonable and flow processing is not affected by the increased number of VTS.

VI. Conclusion

In this paper, we introduced SD-NMS, a novel software-defined architecture enabling multi-tenancy scalable, flexible and autonomic isolation for virtual networks. Our architecture aims at automating the instantiation of a virtual infrastructure while automatically deploying the required security mechanisms to enforce the network isolation between different tenant VTSs. This deployment is driven by cloud tenant's global isolation policy, and thus covers all resources. Our approach demonstrates the simplicity and the feasibility of SDN slicing technique. The flexibility gained through this approach helps to adapt the network dynamically to both unforeseen and predictable changes in the network. It offers the possibility to run multiple slices within the same logical switch without performance degradation.

Future work will be an extension for our paper to demonstrate the simplicity of VTSs scalability which requires more extensive testing that the experiment reflected in this article. Additional testing of tenant's VMs and VTSs migration is in fact part of challenging future work.

Acknowledgment

The authors would like to thank NSERC, Ericsson, Inocybe Technologies and MITACS for supporting the Green Sustainable Telco Cloud Project.

References

[1] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, “Flowvisor: A network virtualization layer,” 2009.

[2] A. Tootoonchian and Y. Ganjali, “Hyperflow: A distributed control plane for openflow,” in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*. USENIX Association, 2010, pp. 3–3.

[3] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, “Onix: A distributed control platform for large-scale production networks.” in *OSDI*, vol. 10, 2010, pp. 1–6.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[5] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, “A survey of autonomic communications,” *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 223–259, 2006.

[6] S. Cabuk, C. I. Dalton, K. Eriksson, D. Kuhlmann, H. V. Ramasamy, G. Ramunno, A.-R. Sadeghi, M. Schunter, and C. Stüble, “Towards automated security policy enforcement in multi-tenant virtual data centers,” *Journal of Computer Security*, 2010.

[7] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “Nox: towards an operating system for networks,” *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 105–110, 2008.

[8] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker, “Applying nox to the datacenter.” in *HotNets*. Citeseer, 2009.

[9] O. Krieger, P. McGachey, and A. Kanevsky, “Enabling a marketplace of clouds: Vmware’s vcloud director,” *ACM SIGOPS Operating Systems Review*, vol. 44, no. 4, pp. 103–114, 2010.

[10] M. Sridharan, M. Pearson, I. Ganga, G. Lin, P. Thaler, C. Tumuluri, A. Greenberg, K. Duda, and Y.-S. Wang, “Nvgre: Network virtualization using generic routing encapsulation,” 2013.

[11] M. Rosenblum, “Vmware’s virtual platform^{TM},” in *Proceedings of Hot Chips*, 1999, pp. 185–196.

[12] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright, “Vxlan: A framework for overlaying virtualized layer 2 networks over layer 3 networks,” *draftmahalingam-dutt-dcops-vxlan-01. txt*, 2012.

[13] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, “A security enforcement kernel for openflow networks,” 2012.

[14] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, “Fresco: Modular composable security services for software-defined networks,” in *To appear in the ISOC Network and Distributed System Security Symposium*, 2013.

[15] E. L. Fernandes, “nox13:oflib,” https://github.com/CPqD/nox13:oflib.

[16] Z. L. Kis, “Openflow 1.1,” https://github.com/TrafficLab.