Domain Isolation in a Multi-Tenant Software-Defined Network

Alireza Ranjbar*, Markku Antikainen†, Tuomas Aura†
*NomadicLab, Ericsson Research, Finland *†Aalto University, Finland
alireza.ranjbar@ericsson.com, markku.antikainen@aalto.fi, tuomas.aura@aalto.fi

***Abstract*—Software-Defined Networking (SDN) has evolved as a new networking paradigm to solve many of current obstacles and limitations in communication networks. While initially intended mainly for single-domain networks, SDN technology is going to be deployed also to large cloud-based data centers where several customers, called tenants, share network resources. In a multi-tenant environment, the SDN technology allows the customers to have higher level of control over the available network resources. However, as the underlying network elements and control logic are shared between multiple tenants, the isolation between tenant domains becomes an important factor in the design of all multi-tenant solutions. In this paper, we propose a scalable system architecture based on OpenFlow and packet rewriting that provides the isolation and controlled sharing between tenants while enabling them to have control over their assigned resources. The architecture addresses different facets of isolation in a multi-tenant network including traffic, address space, and control isolation. Our solution improves on previous ones by putting special emphasis on inter-tenant communication, e.g. on subcontractor relations in cloud services. The evaluation of the prototype indicates that our solution puts only a small performance overhead on forwarding in a shared network.**

***Keywords*-Software-Defined Networking, Domain Isolation, Packet Rewriting, Inter-Tenant Communication, Policy**

I. Introduction

A long-term trend in networking has been towards increasing the sharing of the underlying network infrastructure. This development has been attested by the growing demand for cloud based services in which several tenants share both computing and network resources in a scalable manner [1]. The benefits of multi-tenancy are well known: it makes network maintenance more effective, improves hardware utilization, and reduces the operational costs [2]. With the advent of Software-Defined Networking (SDN), multi-tenant networking has been taken one step further. By exposing an abstraction of the network resources to the tenants through a logically centralized controller, the tenants can gain better control over their own network traffic and routes [3].

While giving tenants more control over their networking resources is tempting, the increased delegation also creates challenges for isolation between the tenants. The most obvious isolation requirement is that network traffic should not leak between the tenants. This alone, however, is not enough: the tenants should be unable to influence each other in any way. For example, the tenants' address spaces should be private and they should be unable to affect each others' performance significantly.

Isolation, however, is not the only desired property in a multi-tenant network. In fact, complete tenant-isolation is typically not even hoped for. There are many scenarios where the tenants may wish to share services with each other without exposing the services to the Internet. It is estimated that this kind of inter-tenant traffic comprises 10% to 40% of today's multi-tenant data center traffic, and the value is expected to increase [4]. Thus, an effective multi-tenant networking architecture should provide both strict isolation for the tenants and the possibility to engage in controlled inter-tenant communication.

This paper was motivated by the lack of a suitable balance between isolation and sharing in the existing multi-tenant solutions. To tackle this challenge, we have designed a system architecture that builds on existing principles and covers the most important isolation and sharing aspects of a multi-tenant network. We consider several different isolation requirements: *traffic isolation*, *address space isolation*, and *control isolation*. The traffic isolation prevents any data-packets from leaking between the tenants while the address space isolation allows the tenants to choose their end-host IP and MAC addresses independently from each other. Control isolation enables the tenants to control and configure their network without affecting other tenants.

Unlike previous proposals that strictly segregate the tenants, our solution enables them to interact with each other and with external resources outside a shared network. Tenants can advertise their services to each other, agree on using each other's services, and set access-control rules for the inter-tenant communication. Network service providers are naturally keen to see such service ecosystems developing within their networks, and the component services offered locally at the same provider network will have performance advantage over those shared over the global Internet.

We implement the isolation by assigning end-hosts into the corresponding tenant domains and tagging the initiated flows in the network. That is, the ingress edge switch takes

This work was supported by TEKES as part of the Cyber Trust program of DIGILE (the Finnish Strategic Center for Science, Technology and Innovation in the field of ICT and digital business). Markku Antikainen was supported by a grant from the Nokia Foundation.

care of most of the policy enforcement and embeds the domain label into the packet headers. Moreover, we have tagged the flows with a routing label, which increases the scalability of the architecture in large multi-tenant networks. The separation of edge and core also makes it possible to implement the architecture on the existing OpenFlow-enabled devices. This is important because the current OpenFlow hardware switches support only relatively small flow tables, which also limits the number of forwarding rules that can be used to enforce isolation policies. Also, this approach is in accordance to the cloud-based data center architecture in which the end-hosts are connected to the software switches in server machines.

We implemented our solution on the OpenDaylight SDN controller and evaluated the prototype in terms of functional correctness and latency on the software switches at the edge of the network. The results show that our approach puts only a very small overhead for forwarding.

The rest of the paper is organized as follows. We start with an overview of the system and design challenges in section II. The main technical contents of the paper about the isolation and forwarding process are presented in section III. Section IV describes the prototype implementation and its evaluation. Section V provides a discussion of the results, and section VI compares our work with the literature. Finally, section VII concludes the paper.

II. System Overview

The following sections provide an overview of the key concepts and design challenges related to multi-tenancy in software-defined networks.

A. Key Concepts and Components

A *software-defined network*, for the purposes of this paper, consists of forwarding nodes, i.e. switches, which forward packets based of relatively simple rules, and of a controller, which makes the routing decisions on behalf of the switches. This architectural model of our work is based on OpenFlow.

The *service provider* is in control of all network resources and is responsible for managing the infrastructure. It is also responsible for configuring the SDN controller and other network elements (namely the switches and gateways). Moreover, the service provider should register tenants to the network by creating accounts and assigning network resources to them.

A *tenant* is typically a customer who rents network resources or receives them as part of some broader service package. Tenants are able to configure their networks independently. The tenants can, for instance, assign subnets for their end-hosts, create policy groups and forwarding policies for their traffic, and advertise their services to other tenants in the shared network.

Tenant's network domain, or simply *domain*, is one tenant's view of the network. It defines the boundaries of the tenant's network, the resources allocated to it, and the tenant's own settings and policies. The domain also includes the end-hosts that are connected to the tenant's network through edge switches. The domain is mapped to the underlying network resources of the service provider, but the tenant does not need to be fully aware of this mapping. In our architecture, there exists exactly one domain for each tenant.

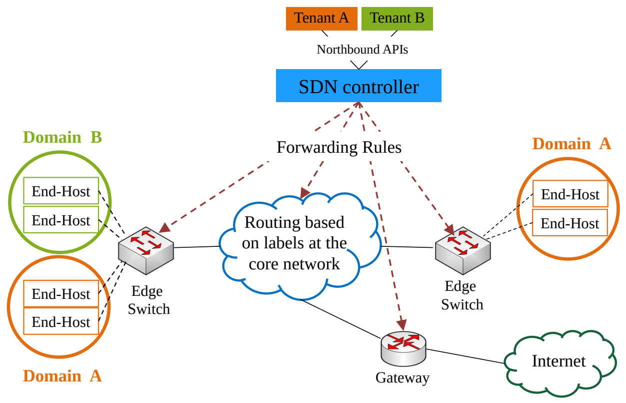


Figure 1: System architecture

Figure 1 illustrates the architecture of the service provider's network, into which we integrate our isolation solution. Tenants and the service provider access the SDN controller through the northbound APIs. The service provider uses them for configuring the network resources and adding or removing tenants. The tenants use the northbound APIs to create subnets, to define policies, and to advertise their inter-tenant services to other tenants. The controller controls the network switches and gateways through OpenFlow. We distinguish between the core network and the edge. The latter comprises the access switches to which end-hosts are connected and the gateways that connect the service provider's network to external networks.

B. Design Challenges and Solutions

In the following, we discuss the main challenges faced in the design of the system architecture.

Address space isolation vs. access between tenants: As already mentioned, we wish to provide address space isolation between tenants while, at the same time, allowing controlled communication between them. However, address space isolation means that the tenants make overlapping use of the private IP address space, which complicates the communication between the tenants. Some solutions (such as NetLord [1]) allow inter-tenant communication only through public IP addresses. This approach, however, is not tempting due to the extra overhead of transmitting the flows through the gateways and the shortage of public IPv4 addresses. Also, public IP addresses may be used as anycast addresses that are routed to multiple locations globally, while we specifically want to enable local sharing within the one operator's network. We solve these problems by tagging the flows of the overlapping address spaces on the same underlying network and by reserving some private addresses for inter-tenant services.

Fine-grained policy enforcement vs. forwarding table size and switching speed: A palatable architecture should support very fine-grained isolation and sharing policies. However, the existing SDN hardware switches can only store a limited number of forwarding rules. Software switches, on the other hand, do not have this limitation. For instance, Open vSwitch can support up to 200k forwarding rules [5]. Inspired by previous solutions [6], [7], we solve this requirement by treating the edge and core of the network separately. In our system architecture, the isolation is enforced at the edge of the network using software switches while at the core network, the use of hardware switches provides efficient and fast forwarding.

Address resolution: The broadcast mechanism in Address Resolution Protocol (ARP) for discovering the MAC addresses is not scalable enough for software-defined networks where there are no clear boundaries between layer-2 segments and which may often host large numbers of virtual machines. Moreover, ARP is not straightforward to apply in multi-tenant environments where different tenants may use overlapping IP addresses. We modify the ARP handling in our solution in such a way that it does not rely on broadcast ARP messages.

III. Isolation and Forwarding

This section explains in detail the isolation and forwarding mechanisms in our solution. We start by explaining the different communication patterns and then continue to the more technical sections on addressing and labeling, domain identification, policy enforcement, and forwarding.

A. Communication Patterns

As has already become clear, we differentiate between three communication patterns: *intra-tenant*, *inter-tenant*, and *external*. The intra-tenant and external communication are familiar everyday patterns, while inter-tenant communication is specific to multi-tenant networks.

The intra-tenant communication limits the scope of flows to a single tenant network. That is, the destination domain of a flow is the same as the source domain. Typical intra-tenant communication in a data-center network would be, for example, access from front-end to back-end servers. For reliability and usability reasons, we allow the intra-tenant communication by default and leave it to tenants to turn on access control between their own end-hosts.

The external communication happens between a tenant and a resource outside the service provider's network. Typical external communication patterns are access from end-hosts in the tenant's network to the Internet and access

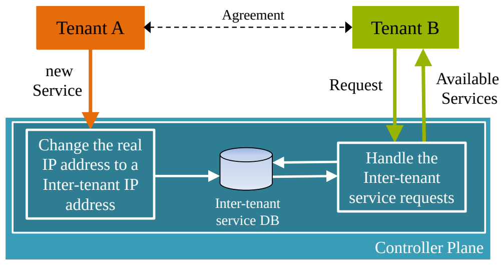


Figure 2: Advertising inter-tenant services

from entities in the Internet to the end-hosts in the tenant's network. The default policy for external communication is a stateful firewall-like policy that prevents both inbound or outbound connections.

Finally, the inter-tenant communication pattern enables different tenants to communicate with each other within the service provider's network. Typically, an end-host in one tenant's network accesses a service in another tenant's network. On the packet level, this flow type can be identified by the different source and destination domains. The inter-tenant communication must be explicitly allowed in the policies defined by both tenants.

Each tenant that offers inter-tenant services should advertise them. This is done using the controller's northbound API. While tenants can choose any port number for their services, the IP address for the advertised services will be allocated from a special pool. Figure 2 shows an example of the service advertisement process. At first, tenant A has agreed to offer a service to tenant B, which may or may not involve payment and legal contracts. The next step is for tenant A to advertise the service using the northbound API. Tenant B can then request reachability information from the service provider's controller using a unique service identifier. The reply to this request contains the IP address and port number that tenant B can use to access tenant A's service. However, if access to the service is not allowed by tenant A's policy, the reply will be empty, and access to the service's IP address is also prevented on the forwarding layer.

B. Addresses and Labels

Next, we need to explain the use of IP addresses and other packet header fields in our solution.

To enable inter-tenant communication, we split the largest private IPv4 address block (10.0.0.0/8) into two parts. The subnet 10.0.0.0/9 is reserved for each tenant to be used for its internal network. The tenants are allowed to assign overlapping addresses in this range, and end-hosts with only such an address will be strictly isolated from the other tenants. The address range 10.128.0.0/9, on the other hand, is reserved for inter-tenant communication. The tenants planning to provide inter-tenant services will be given one or more IP-addresses from this range, and end-hosts that access inter-tenant services are dynamically allocated an address-port pair. These identifiers are unique within the service provider's network, across tenants. External communication naturally requires the use of public IP addresses, which are routable beyond the service provider's network. As usual, these may be allocated statically or dynamically and owned by the tenant or by the service provider.

The use of inter-tenant and public IP addresses is almost unnoticeable to the end-hosts. The end-hosts always retain their own intra-tenant IP address from the range 10.0.0.0/9, which they receive from DHCP, and use it for all network communication. The controller keeps track of the public and inter-tenant addresses and the edge switch near the end-host maps the addresses as needed (similar to a NAT). The only situation where the end-host sees an inter-tenant IP address is when it connects to the address that was advertised by an inter-tenant service. The address mapping and forwarding mechanisms will be discussed in more detail in section III-E.

The layer-2 MAC addresses are not used for end-to-end communication but, instead, as labels for isolation and routing. Since the IP address alone is not sufficient to identify the destination end-host for intra-tenant communication (because of overlapping IP addresses), we use the source MAC address as the *domain label*. Each domain has a unique 48-bit domain label, allocated by the service provider. The domain label identifies the set of virtual (and possibly physical) edge switch interfaces where the tenant's end-hosts are located.

Forwarding in the core network is based on the *routing label*. Inspired by [8], we use the destination MAC address as the packet-header field for the routing label because it guarantees the most efficient forwarding behavior on all types of switches and because it enables interoperability with legacy (non-SDN) switches. The routing label usually identifies the egress edge switch. The architecture, however, allows the controller to assign different routing labels to different end-to-end paths that lead to the same egress switch. This may be necessary for advanced routing features such as load balancing and multi-path flows.

The end-hosts do not need to be aware of the domain or routing labels and in most cases will not see them. The network, on the other hand, makes no use of the end-host MAC addresses. The response to all ARP requests is the controller's MAC address, which is the safest choice for a constant address. In this way, every entry in the end-hosts' ARP tables will have the same value. In a case this causes problems for a domain, it is possible to assign virtual MAC addresses to its end-hosts.

We have selected the packet fields for the labels for maximum scalability and performance. The edge switch in data center networks is often a software switch, which means that it is able to handle the rewriting of labels and MAC addresses without significant performance penalty.

C. Domain Identification

In order to label packets and enforce the isolation policies, it is necessary to determine the domain for each data flow. In our architecture, each domain is associated with a set of input ports of the edge switches. Thus, it is possible to simply use the switch input ports for identifying the corresponding domain. When the controller plane receives a forwarding request (packet-in message in the OpenFlow protocol) from a switch, the request includes information about the switch input port. This information about the switch port is used to decide the source domain. The destination domain, on the other hand, is determined based on the destination IP address. Since the architecture distinguishes intra-tenant, inter-tenant and external communications, the controller needs to check to which IP range the destination IP address belongs. The domain identification procedure is depicted in Algorithm 1. Note that there is a separate database table for mapping public IP addresses to the tenants who have been allocated such addresses, but unrecognized public addresses are mapped to the domain identifier zero, which means the Internet and other external networks.

Algorithm 1 Domain identification algorithm

```
1: $dom_{src} \leftarrow domainPortDB.get(port_{input})$
2: if $ip_{dst} \in$ 10.0.0.0/9 and $dom_{src}$ != 0 then
3:     $dom_{dst} \leftarrow dom_{src}$
4: else if $ip_{dst} \in$ 10.128.0.0/9 and $dom_{src}$ != 0 then
5:     $dom_{dst} \leftarrow domainInterDB.get(ip_{dst})$
6: else if $ip_{dst} \in IP_{public}$ and $dom_{src}$ == 0 then
7:     $dom_{dst} \leftarrow domainExtDB.get(ip_{dst})$
8: else
9:     $dom_{dst} \leftarrow 0$
10: end if
```

Once the source and destination domains of a new flow have been identified, the policies from these domains are used to decide whether the flow is permitted and how it should be forwarded.

D. Policy Enforcement

The policy in our solution is defined as a table of rules, which are matched against the packets to determine the correct action. As shown in Table I, each policy contains three parts: *flow type*, *match fields*, and *action*. The flow type defines the communication patterns to which the policy applies; the possible values are *intra-tenant*, *inter-tenant*, and *external*. The *match fields* define a set of criteria against which the packet headers are matched. Our solution supports the same match fields as OpenFlow. Finally, *action* defines what to do with the matching flows. Our basic approach is to allow all flows that match any rules and drop all non-matching flows. However, the action attribute may contain additional information about how the flow should be routed or about its quality-of-service requirements.

Algorithm 2 shows the procedure for verifying that a flow complies with the policies. For intra-tenant communication,

Table I: Sample policy definition

Flow type	Match field	Action
Intra-tenant	Src IP: 10.0.0.0/9 Dst IP: 10.0.0.0/9	Allow (best effort)
External	Src IP: 10.0.0.0/9 Dst IP: *	Allow (best effort)
Inter-tenant	Src IP: 10.0.0.0/9 Dst IP: 10.128.1.1	Allow (best effort)

since the source and destination domains are the same, we check the data flow against the source domain's policy and apply the action from there. For inter-tenant communication, the data flows must be allowed by the policies of the source and destination domains (both tenants), and the action specified in the policy for the source domain should be equal to the action in the policy of the destination domain. In case of external communication, for outbound flows, the policies of the source domain should allow the flow and, for inbound flows from the Internet, the flows are verified based on the polices for the destination domain.

Algorithm 2 Flow Verification Algorithm

```
    Intra-tenant communication
 1: if $dom_{src}$ is equal to $dom_{dst}$ and are not 0 then
 2:     for all intra-tenant policies do
 3:         if flow is matched with $pol_i.rules$ then
 4:             $pol_i.action$ ← allow intra-tenant flow
 5:             return $pol_i.action$
 6:         end if
 7:     end for
 8: end if
    Inter-tenant communication
 9: if $dom_{src}$ is not equal to $dom_{dst}$ and are not 0 then
10:     for all inter-tenant policies in $dom_{src}$ do
11:         if flow is matched with $pol_i.rules$ then
12:             $pol_i.action_{src}$ ← allow inter-tenant flow
13:             $break$
14:         end if
15:     end for
16:     for all inter-tenant policies in $dom_{dst}$ do
17:         if flow is matched with $pol_i.rules$ then
18:             $pol_i.action_{dst}$ ← allow inter-tenant flow
19:             $break$
20:         end if
21:     end for
22:     if $pol_i.action_{src}$ is equal to $pol_i.action_{dst}$ and they are not 0
    then
23:         return $pol_i.action_{src}$
24:     end if
25: end if
    External communication
26: if $dom_{src}$ or $dom_{dst}$ is 0 then
27:     for all external policies do
28:         if flow is matched with $pol_i.rule$ then
29:             $pol_i.action$ ← allow external flow
30:             return $pol_i.action$
31:         end if
32:     end for
33: end if
```

E. Forwarding

Depending on the result of flow verification, the controller configures the switches in the network for dropping the flows at the edge of the network or forwarding the flows through the edge and core network.

Forwarding at the edges:

The edge switches rewrite some packet headers with the necessary labels. The choice of headers depends on the flow type. Figure 3 illustrates the packet rewriting at the edge of the network in the case of intra-tenant and inter-tenant communication. For the intra-tenant communication, the source and destination MAC addresses are simply rewritten with a domain label and a routing label, respectively. The domain label identifies the source tenant's domain. Therefore, this label makes it possible to distinguish between flows of different tenants that have overlapping IP addresses. The routing label, on the other hand, identifies usually the egress edge switch to which the packet will be sent.

The processing of inter-tenant communication at the edge switches is similar to the intra-tenant communication with the difference that also the IP addresses and port numbers are rewritten. The destination IP address and port number, which are from the inter-tenant address block 10.128.0.0/9 are replaced with the destination end-host's intra-tenant IP address and port, which are registered at the controller. Furthermore, the source IP address and port number are rewritten with an inter-tenant IP and port number. If the source node does not yet have such values, the controller assigns them dynamically. Additionally, for intra-tenant and inter-tenant communication, the routing manager installs a forwarding rule to the egress switch. This rule basically just rewrites the destination MAC address field with the MAC address of the destination end-host. The source MAC address is, however, not rewritten at the egress switch. This guarantees that no layer-2 address space information is leaked between different tenants.

Figure 4 shows a similar picture of external communication. In this case, if the flow is *towards* the external network, the edge switch next to the source writes the domain label and the routing label into the packet's source and destination MAC address fields, respectively. These allow the flow to be forwarded to the external gateway. Moreover, the source IP address and port number are set to a mapped public IP address and port number. When the flow reaches the gateway, the gateway changes the destination MAC address to whatever is needed for the next hop. Similarly, when a flow originates *from* the external network, the gateway will set the source MAC address to the destination host's domain label and the destination MAC address to the routing label, which are then used for forwarding the flow through the core network and to the end-host. At the egress edge switch next to the destination, the flow originating from an external network is modified again so that the destination MAC

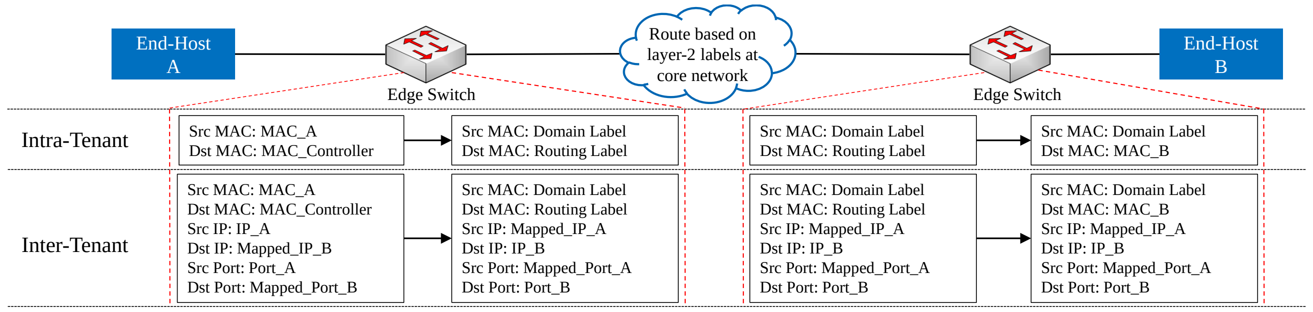


Figure 3: Forwarding at the edge switches for intra-tenant and inter-tenant communications

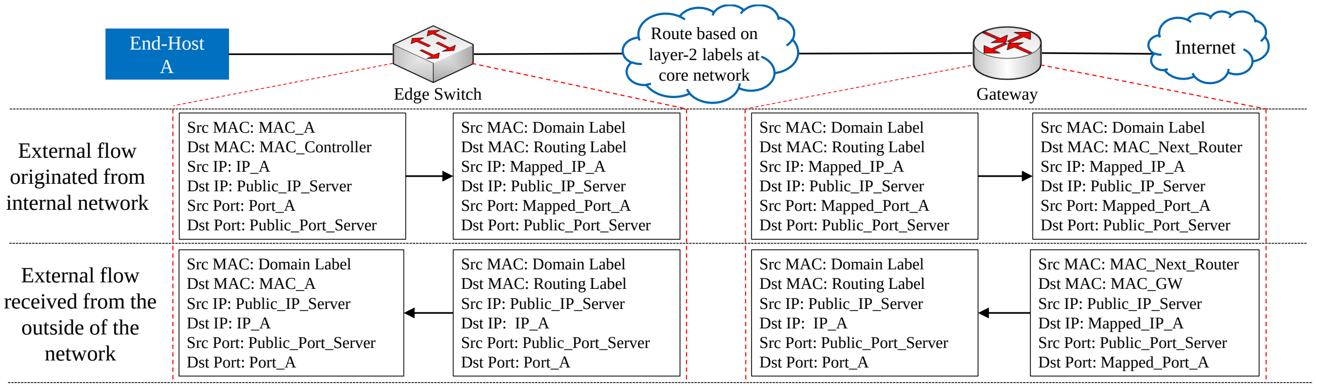


Figure 4: Forwarding at the edge switches for external communications

address is changed to the MAC address of the destination end-host.

Forwarding in the core network:

So far, we have discussed how the packet headers are modified at the software switches at the network edges. Forwarding in the core network, on the other hand, is performed by hardware switches. They perform the forwarding based on the routing label, which is in the destination MAC address field. While the flow verification is triggered reactively at the ingress edge of the network, the controller installs forwarding rules proactively to the end-to-end path through the core network. This is necessary to limit the latency of rule installation for new flows.

Performance violations in forwarding:

There are situations in which a tenant may wish to forward large amount of traffic in a shared network. Considering the limitations in network resources, this situation can affect negatively the performance of other tenants in a shared network. To detect this type of unusual traffic, the shared network is monitored by receiving and analyzing the statistics from the switches at the controller. While it is possible to collect the statistics using the OpenFlow protocol, our prototype uses sFlow for scalability reasons [9].

IV. Implementation and Evaluation

We implemented the prototype on the OpenDaylight controller (Hydrogen release). The internal modules of the controller are shown in Figure 5.

The tenants and service provider configure network settings through the northbound API, which is handled by the service manager module. The main role of the service manager is to manage the domains and to map all of the configuration changes to a particular domain. Moreover, the other modules are able to query domain-related information from the service manager.

The DHCP server statically or dynamically assigns IP addresses to end-hosts in the network. In order to allow overlapping IP addresses, each domain is assigned a separate DHCP pool. The tenant may create subnets of different sizes with their own address pools. After address allocation, the DHCP server stores information about all detected end-hosts, their location in the network topology, and which domain they belong to.

The ARP handler takes care of ARP requests. When an edge switch receives an ARP request, it forwards the request to the handler instead of broadcasting it. The ARP handler responds to all ARP requests with the MAC address of the SDN controller.

The isolation manager is responsible for enforcing the isolation policies in the network. All new host-initiated flows

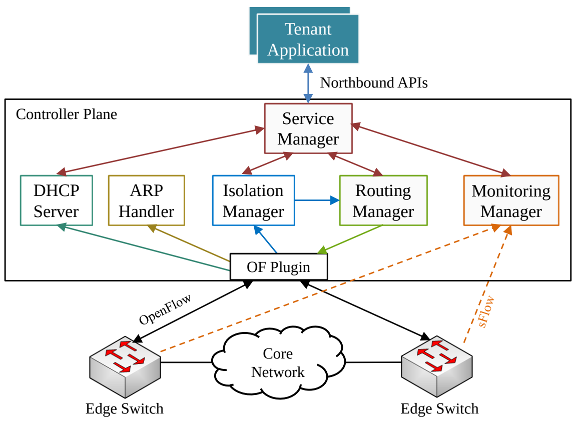


Figure 5: Controller-plane components

that arrive to the edge switch are sent to the isolation manager as flow requests. The approved flow requests are then delivered to the routing manager, which determines a path through the core of the network. The routing manager then installs the forwarding rules to rewrite the packet headers at the edge switches and to forward the flows through the core network. Any performance violations in forwarding process are detected by the monitoring manager, which is responsible for gathering statistics from the switches with the sFlow protocol.

The implemented prototype consists of two bundles. One of them implements the northbound APIs while the other bundle implements the rest of the modules in the system architecture. All modules are defined in about 6000 lines of Java code (without comments). Additionally, for implementing the monitoring features, we used sFlow-RT [10], which provides the real-time information about the network performance.

A. Functional correctness

We used Mininet v2.1.0 [11] to test the functionality of the prototype. While testing in an emulated environment does not prove the correctness of the principles or implementation, experience shows that such testing will often detect flaws that escape theoretical analysis or go undetected in experimental deployment.

A simple test network is illustrated in Figure 6. In this network, we have three tenants A, B and C. Each tenant has two end-hosts connected to different edge switches. End-hosts in the emulated network access the external network using a gateway. Both the edge switches and the gateway are implemented on Open vSwitch.

To make the test more challenging, we initiated overlapping flows with matching source IP and MAC addresses and port numbers for all the different flow types. For intra-tenant communication, we created overlapping TCP connections between end-hosts of the same tenant. For inter-tenant communication, we offered a service on end-host C-2 and created overlapping flows from end-host A-1 and B-1 to the end-host C-2. For external communication, we opened TCP connections to a server outside of the network. The architecture and implementation was observed to handle all these cases correctly.

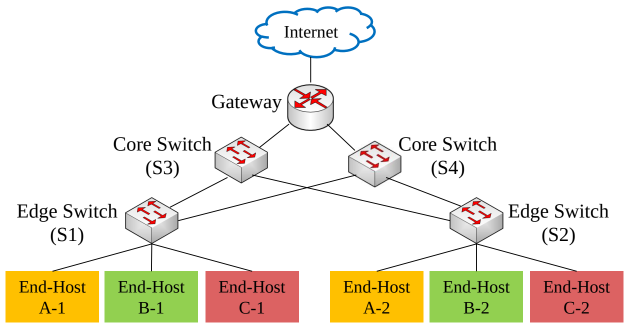


Figure 6: Testbed network

B. Performance

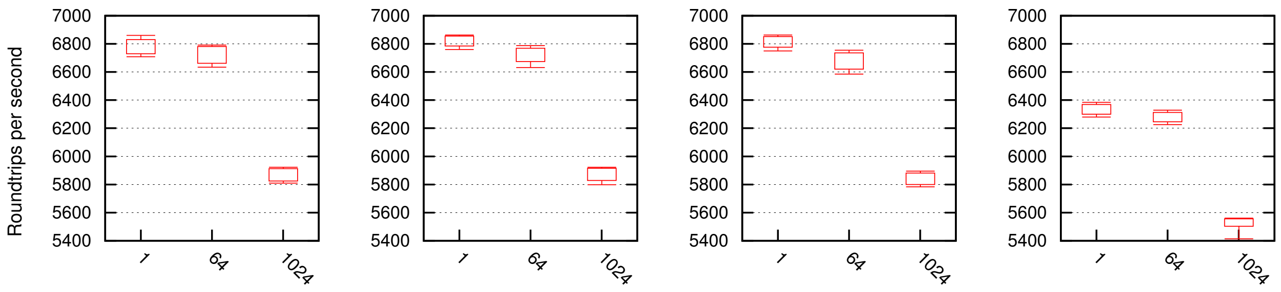
One of the main concerns about the domain isolation approach described above was the performance of packet rewriting in the software switches at the network edge.

We ran a test to check the latency of packet rewriting. The test configuration consists of two 64-bit Ubuntu 14.04 machines, each with a 2.66 GHz quad-core Intel Core 2 CPU and 8 GB RAM. Both machines run the Open vSwitch v2.0.2. The software switches are connected to each other through a 1Gbps Ethernet switch. Similar to the network in Figure 3, we ran flows from one machine to the other machine and measured the performance of the software switch in the different scenarios.

For the measurements, we used the TCP_RR (TCP request-response) test in Netperf v2.6.0 [12], which measures the number of transmitted and received TCP packets between two machines. We did this measurement for the plain forwarding, intra-tenant, and inter-tenant communication. Additionally, to have a point of comparison, we ran the test with VXLAN encapsulation, which is one of the most popular domain isolation approaches in data centers that can be implemented with OpenFlow.

The measurements were repeated for different packet sizes and 100 times for each size. The results can be seen in Figure 7. It is clear that, for intra-tenant and inter-tenant communication, the performance of our isolation solution in different communication patterns is comparable to plain forwarding. However, the performance of VXLAN is considerably lower in the same test.

One reason for the performance degradation of encapsulation methods like VXLAN is the overhead caused by larger packets [1]. The VXLAN packets might be larger than the default MTU (1500 bytes). The low performance of VXLAN might also be caused by the small socket buffer size or buffer descriptor size on network interface cards, as



(a) Plain Forwarding (b) Intra-Tenant (c) Inter-Tenant (d) VXLAN

Figure 7: Forwarding efficiency with no packet rewriting (a), intra- and inter-tenant routing (b-c), and with VXLAN encpasulation (d). The candlestick shows the 5th-to-95th percentile range and the extreme values.

suggested in [1]. However, the results still indicate that the approach proposed in this paper performs at least as well or better than VXLAN.

V. Discussion

In this section, we will compare the presented solution to the design goals and try to set it in the wider context.

Domain isolation: The main goal of this work was to develop a domain-isolation solution for multi-tenant networks while also allowing controlled access between the tenants.

Our solution provides *traffic isolation* between the tenants by verifying all flows at the ingress edge switch of the network against the policies defined by the tenants themselves. Before forwarding any traffic between the end-hosts or through the core network, the edge switch (under supervision of the SDN controller) checks that only legitimate flows will be forwarded. The end-hosts are associated with domains, and all packets in the network are tagged with domain labels to ensure that they are forwarded to only authorized destinations.

Our solution achieves partial *address space isolation* in layer-3 because we allow the tenants to expose services to each other. This is, of course, intentional, and a tenant's end-hosts can remain completely isolated from other tenants by using only addresses from the block reserved for intra-tenant use. The rewriting of IP addresses at the edge switches ensures that the internal addresses of a tenant network are never leaked to outsiders. We implement complete layer-2 address space isolation in the sense that the tenants can configure freely the MAC addresses of their end-hosts and those addresses are never propagated to any other hosts.

Our system architecture allows tenants to control their own networks, while also providing a high degree of *control isolation*. The tenants are able to set policies for traffic forwarding, create subnets, and assign IP addresses to the end-hosts in their networks by configuring the DHCP server. The tenants are limited to their own networks and they are not able to affect the global administrative configuration or the configurations made by other tenants in the multi-tenant network.

Controlled access between tenants: As a balance for the isolation, our solution supports controlled inter-tenant and external communications. The tenants are able to advertise services within the network, sign contracts outside the virtual world, and open the services for access by other tenants. Access to the Internet and other external networks is naturally also allowed, but we put special emphasis on inter-tenant service access within the same multi-tenant network. The controlled service access between tenants is an attractive proposition for the network service provider. In a data center, the service provider will naturally want to encourage the growth of a service ecosystem that takes advantage of the efficient and secure local access between the tenants.

Deployability and Scalability: One of the design principles of our solution was to use inexpensive commodity hardware switches for the packet forwarding at the core network. To avoid scalability issues in the core network, we embed a routing label in the destination MAC address to aggregate and forward a set of flows through the network. This makes the forwarding rules at the core network simple and also provides compatibility with the legacy switches [8].

The scalability issue in the core network is particularly burning because the current OpenFlow hardware switches suffer from small flow table sizes that restrict the number of forwarding rules simultaneously on the device. Unlike the hardware switches, software switches such as Open vSwitch are able to support up to 200k microflows [5]. As a result, the use of software switches allows us to implement fine-grained isolation for all flows that originate at the edge network and to keep the core network simple for forwarding operations. When it comes to traffic originating from the Internet, a gateway switch/router is responsible for forwarding and isolation at the edge of the network that is connected to the Internet. On one hand, the gateway should implement fine-grained isolation policies on the flows received from the Internet and, on the other hand, it should be able to forward large traffic volumes and perform the routing tasks of a gateway router. We follow the approach of [13], [7] in using software for this task.

Ability to interoperate with legacy hardware, which was mentioned above, is important for deployability of the solution as we cannot expect the network operators to replace all their hardware in one go. Fortunately, since our approach is based on simple layer-2 forwarding rules, it can be deployed with mixed SDN and legacy switches. Furthermore, if we look into the future, the use of routing labels will enable the deployment of advanced technologies such as multi-path routing and load balancing between edge switches without changes to the operating principles. Any optimizations that require a large number of diverse paths or packet-processing rules will, however, need to wait until the core SDN switches support sufficiently large forwarding tables.

Scalability of the addressing scheme: It is a continuing concern with IPv4 that the number of addresses is too small for various applications. Even the largest address blocks, such as the class-A block 10.0.0.0/8 that is reserved for private use, are relatively small compared to the number of end-hosts in modern networks. Thus, the IP address allocation was one of the key design decisions in our design process.

Table II presents the number of available IP addresses and port numbers for each flow type. For intra-tenant communication, tenants can use overlapping addresses in the 10.0.0.0/9 subnet, which yields about 8 million addresses for each tenant. Moreover, they are allowed to create internal subnets and implement (virtual) NAT devices between them. The range 10.128.0.0/9 is used for inter-tenant communication between all tenants, and these addresses are allocated by the service provider based on the tenants' needs. Should the number of inter-tenant connections grow, the service provider can further save addresses by allocating only IP address-port pairs for inter-tenant communications. The service provider can make such changes because these addresses are handled by the edge switches under its control, rather than by the end-hosts themselves. For external communication, we have to rely on the existing ways of allocating public IP addresses. The service provider should assign at least one public IP address to each tenant. For outbound connections towards the Internet, a NAT policy can be implemented for sharing public IP addresses.

The other private IPv4 address blocks (172.16.0.0/12 and 192.168.0.0/16) may be used in addition to 10.0.0.0/8, but their combined size is only a fraction of the largest block and, thus, will have no major effect on scalability.

Encapsulation vs packet rewriting: As will be clear from the following section on related work, one of the key technical decisions in our solution was the choice of packet rewriting instead of encapsulation. Encapsulation and tunneling techniques such as STT [14] or VXLAN [15] are quite popular solutions for isolation between tenants in existing multi-tenant networks. However, we believe that the packet rewriting method discussed in this paper brings benefits compared to the traditional encapsulation methods. One of the advantages of packet rewriting is its negligible overhead in forwarding. Based on the result of our evaluation, today's software switches such as Open vSwitch can do the packet rewriting at the same rate as plain forwarding on commodity server hardware. In comparison, the encapsulation methods such as VXLAN or STT create more computational overhead, and they might also cause packet fragmentation problems. Furthermore, the encapsulation techniques do not have any obvious means for supporting inter-tenant communication. Indeed, if tenants want to communicate with each other, we should make sure that the two sides of the communication use different IP addresses. For this purpose, we reserved a subset of the private IP address space for communication between tenants. Last but not least, the packet rewriting enables simple and efficient isolation policy enforcement because it allows us to embed explicit labels in the packet headers. Rewriting the packet headers at the edge of the network, rather than depending on the end-hosts to set them correctly, gives the network complete control over the enforcement of the isolation policies.

Table II: IP addresses and ports for different flow types

Flow type	Available addresses	# IP Addresses	Available ports
Intra-tenant	10.0.0.0/9	2^{23} / tenant	0-65535
Inter-tenant	10.128.0.0/9	2^{23} / all tenants	0-65535
External inbound	Available public IP addresses	≥ 1 / tenant	0-65535
External outbound	Pair of public IP address and port	≥ 1 / tenant	1024-65535

VI. Related Work

There are different ways how multi-tenancy can be implemented with SDN technologies. One way is to place a proxy (hypervisor) between the controller plane and the data plane, which enforces isolation. FlowVisor [16] is a well-known example of this approach which assigns a network slice to each tenant. However, it does not allow the tenants to have full control over their address space and also does not allow inter-tenant communication. OpenVirtex [17] is another proxy-based solution that provides traffic and address space isolation between tenants, but it does not aggregate flows at the core network, and inter-tenant and external communication are not considered.

Another multi-tenant approach is to allow tenants to directly connect their applications to the SDN controller. FlowN [18] implements multi-tenancy by assigning containers to tenants and by tagging each tenant's traffic with a different VLAN tag. Following this approach, Gutz et al. [19]

proposed a solution in which the tenants' applications are compiled on the controller plane and a slice of the network resources is assigned to each tenant. The flows at the data plane are tagged with a unique VLAN corresponding to a tenant. However, VLAN tagging does not scale well for large multi-tenant networks. Recently, Koponen et al. [20] proposed a multi-tenant solution called Network Virtualization Platform (NVP). The multi-tenancy in NVP is solved by creating overlay tunnels between tenants. While NVP provides traffic and address space isolation, the tunneling approach adds higher overhead. Also, NVP does not provide flow aggregation at the core network, which may cause scalability problems.

The solution in [21] allows tenants to connect directly to the switches at the data plane. The network provider configures the network elements using a master controller and assigns a set of network resources to each tenant. For isolating tenants from each other, the traffic of each tenant is tagged with a VLAN label based on the input switch port. However, since this approach allows tenants to directly connect to the network elements, it is not recommended for multi-tenant networks [3].

VII. Conclusion

We have designed a scalable architecture for multi-tenant software-defined networks that balances isolation between tenants with the need to share services in a controlled way. Our solution puts special emphasis on inter-tenant communication in order to enable the development of a service ecosystem among the tenants. Scalability and strong enforcement of isolation policies are achieved by placing the intelligence at the network edge switches, which rewrite the packet headers with the addresses and labels needed for efficient packet forwarding and domain isolation. The evaluation of the prototype implementation indicates that our solution puts only a small performance overhead on forwarding in a shared network. We intend to continue this work by integrating the described architecture with cloud-oriented platforms such as OpenStack [22].

References

[1] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: A Scalable Multi-tenant Network Architecture for Virtualized Datacenters," in *Proceedings of the ACM SIGCOMM 2011 Conference*, pp. 62–73.

[2] C.-P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. 't Hart, "Enabling multi-tenancy: An industrial experience report," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*, Sept 2010, pp. 1–8.

[3] "SDN architecture," Issue 1.0, TR-502, June 2014, ONF, https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf .

[4] A. Sayler, E. Keller, and D. Grunwald, "Jobber: Automating Inter-Tenant Trust in the Cloud," in *Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2013.

[5] B. Pfaff, J. Pettit *et al.*, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, May 2015, pp. 117–130.

[6] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A Retrospective on Evolving SDN," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 85–90.

[7] X. Jin, L. E. Li, L. Vanbever, and J. Rexford, "SoftCell: Scalable and Flexible Cellular Core Network Architecture," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, 2013, pp. 163–174.

[8] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow MACs: Scalable Label-switching for Commodity Ethernet," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, 2014, pp. 157–162.

[9] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments," *Computer Networks*, vol. 62, pp. 122–136, Apr. 2014.

[10] "Sflow-rt," http://www.inmon.com/products/sFlow-RT.php.

[11] "Mininet," http://mininet.org.

[12] "Netperf," http://www.netperf.org/netperf.

[13] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-Defined Internet Architecture: Decoupling Architecture from Infrastructure," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*.

[14] J. G. B. Davie, Ed., "A Stateless Transport Tunneling Protocol for Network Virtualization (STT)," IETF, Internet-Draft, apr 2014, https://tools.ietf.org/html/draft-davie-stt-06.

[15] M. Mahalingam *et al.*, "Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," IETF, RFC 7348, August 2014.

[16] R. Sherwood *et al.*, "Can the Production Network Be the Testbed?" in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, 2010, pp. 1–6.

[17] A. Al-Shabibi, M. D. Leenheer, M. Gerola, and Ayaka, "Openvirtex: A network hypervisor," in *Open Networking Summit 2014 (ONS 2014)*, Santa Clara, CA, Mar. 2014.

[18] D. Drutskoy, E. Keller, and J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," *Internet Computing, IEEE*, vol. 17, no. 2, pp. 20–27, March 2013.

[19] S. Gutz, A. Story, C. Schlesinger, and N. Foster, "Splendid Isolation: A Slice Abstraction for Software-Defined Networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 79–84.

[20] T. Koponen *et al.*, "Network Virtualization in Multi-tenant Datacenters," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, 2014, pp. 203–216.

[21] P. Skoldstrom and K. Yedavalli, "Network Virtualization and Resource Allocation in OpenFlow-based Wide Area Networks," in *Communications (ICC), 2012 IEEE International Conference on*, June 2012, pp. 6622–6626.

[22] "Openstack," http://www.openstack.org.