Traffic Isolation on Multi-Tenant Data Center Networks

Heitor Moraes
Universidade Federal de Minas Gerais
motta@dcc.ufmg.br

Marcos A. M. Vieira
Universidade Federal de Minas Gerais
mmvieira@dcc.ufmg.br

Dorgival Guedes
Universidade Federal de Minas Gerais
dorgival@dcc.ufmg.br

ABSTRACT

To satisfy demanding clients and offer features comparable to the competition, infrastructure-as-a-service providers (IaaS) need fast, flexible and easily configurable local networks. OpenStack is one of the most well known open IaaS platforms. Although OpenStack meets most needs of a IaaS platform, its virtualized network implementation still lacks flexibility to support isolation on a multi-tenant network environment.

In this work, we developed an application to provide traffic isolation on a multi-tenant data center network that works together with OpenStack, offering a virtualized network environment that follows the SDN paradigm.

VLANs, GRE, and MPLS require extra packet fields to encapsulate packets and provide traffic isolation. Moreover, those approaches have a limited network segment space. For instance, VLANs are subject to scalability limitations resulting from space limitations since only 4000 VLANs are allowed. Our approach does not require additional packet fields neither has small segment space.

By applying the Software Defined Networks paradigm, the isolation process through packet header re-writing is utilized without requiring any changes to the virtualized network environment or any special hardware.

We evaluate our application in real scenarios measuring latency, bandwidth, isolation. The results validate our application for Multi-Tenant data centers.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: [Network Architecture and Design , Network Communications]; C.2.4 [**Distributed Systems**]: Network operating systems, SDN

General Terms

Computer Networking

ACM SIGCOMM SYMPOSIUM ON SDN RESEARCH '15 Santa Clara, California USA

Keywords

Software Defined Networking, Data center, Traffic Isolation

1. INTRODUCTION

Infrastructure as a Service (IaaS) providers have a growing demand for solutions to increase control of the resources offered to its customers. Network topologies created in these environments usually have multiple clients with multiple virtual machines and interconnections, requiring more flexibility and isolation capabilities from the supporting infrastructure.

Because each customer has specific demands, the IaaS platform needs to facilitate the construction of arbitrary virtual network topologies that meet the requirements, without loss of performance. In these environments, ensuring traffic isolation between customer networks is essential, since the client traffic must not affect others.

Software Defined Networking (SDN) is a paradigm that can increase the network flexibility of IaaS platforms. In this type of solution, changes in cloud structure are sent to the SDN central controller and this entity is responsible for orchestrating the necessary changes to the network configuration.

In this work, we developed an application called DCPortalsNG to provide traffic isolation on a multi-tenant data center network. This application works together with OpenStack, one of the most well known open IaaS platforms. DCPortalsNG offers a virtualized network environment that follows the SDN paradigm. Through communication between the application and the deployed module, IaaS platform settings are translated into instructions to be applied in SDN environment. Our solution only uses public APIs defined by the platforms, being more likely to be adopted. Also, our solution remains consistent with IaaS systems because it can be easily updated whenever changes occur in the APIs.

Due to the size of data center networks, it is common to split these networks into smaller independent areas, ensuring isolation between them. One of the most common way is through the use of VLANs [9], but it has serious limitations. VLANs are subject to scalability limitations since only 4096 VLANs are allowed.

In addition to maintaining all the flexibility offered by IaaS platform, other premise of this work is that the developed solution should work in networks physical composed of extremely simple equipment.

The main contributions of our work are the following. The development of an application, called DCPortalsNG, which is capable of providing isolation of customer networks in a

cloud virtualization environment. That capacity is achieved requiring no special features from routing network element. This approach also relieves pressure from their MAC address tables.

Moreover, in addition to the isolation capabilities, the application implements a component that works as a bridge between the cloud virtualization environments and an SDN controller. In this solution, the component is responsible for managing all networks implemented in the environment, making the mappings and necessary settings within the SDN network.

This paper is organized as follows. In Section 2, we introduce the basic concepts and related work. In Section 3, we detail how the system was designed and implemented. In Section 4, we validate our system through experiments and analyse the results. In Section 5, we conclude this work and present possible future projects.

2. RELATED WORK

OpenStack is a set of complementary applications that work to provide a virtualized infrastructure datacenter. This environment provides most of the characteristics that would be expected of a real, such as server allocation, network definition, access control,firewalls, high availability, etc. OpenStack is one of the most well known open IaaS platforms. Although OpenStack meets most needs of a IaaS platform, its virtualized network implementation still lacks flexibility to support isolation on a multi-tenant network environment.

By providing all the infrastructure of a datacenter as a service, OpenStack allows users to define arbitrary networks for interconnection of your virtual machines and, if they wish, also to the Internet. These networks are, usually, virtual networks that represent the interconnection of virtual machines of each user. Responsibility for ensuring the construction of these networks is the *Neutron* [16] component that acts as an abstraction layer, providing an API to other OpenStack components and interacting with the applications that really make the necessary settings for creating networks.

In order to increase the flexibility and facilitate the integration of new technologies, the *Neutron* communicates with the tools through internal mini-applications, called *plugins*, which can be easily developed and extended integrated into the component.

With the emergence of several *plugins* for *Neutron*, the developers realized that many had similar characteristics. Thus, for simplicity and to avoid the development of diverse implementations of persistence layer, they created a special *plugin* called *Modular Layer 2* plugin (ML2) [15]. Differently of *plugins* developed for specific technologies, ML2 allows simultaneous use of two different network layer technologies such as VLANs, VxLANs [19], and encapsulation. This variety of technologies is common found in large datacenters, making the *plugin* extremely useful for that environment.

VLANs, GRE [6], and MPLS [18] require extra packet fields to encapsulate packets and to provide traffic isolation. Moreover, those approaches have a limited network segment space. For instance, VLANs are subject to scalability limitations resulting from space limitations since only 4096 VLANs are allowed. Our approach does not require additional packet fields neither has small segment space. IaaS might require thousands or even millions of clients.

Another possible isolation solution through encapsulation is Q-in-Q [10], which also makes use of VLAN header, but it doubles the header. This solution allows a much larger amount of isolated networks to run in parallel, but depending on data center size, it may still be insufficient. In addition, as with VLANs, the number of virtual machines in the environment can be so great that the switches will have trouble keeping all network addresses (MAC) in its memory.

One of the first techniques for virtual network isolation was developed by the HP Labs group[3]. They introduced the Trusted Virtual Domains (TVDs) concept, which are isolated network section, independent of the topology. To ensure isolation, they developed an internal module for virtual machines, which is responsible to process all traffic related to the isolation. They compared two isolation techniques, Ethernet encapsulation with EtherIP and VLAN tagging. Although this work has similar goal, their approach has scalability limitations and require intrusive modifications at the virtualization system. They also presented a comparative study at [2].

DCPortalsNG is based on the SDN paradigm to solve the problem of isolation. In [17], the feasibility of such solutions is discussed, but no concrete solution is presented. Two applications of data center using the NOX controller were published previously, but they focus on implementing new network architectures and traffic control ([20], [8]). Unlike these solutions, DCPortalsNG does not require equipment that supports OpenFlow in the network core, and focus only in traffic isolation.

The system developed by [14] adopted a technique similar to our work to reduce the size of routing tables in switches inside datacenter. The solution was to encapsulate packets leaving virtual machines, creating a new packet header, addressing them to the final switch in which was located the destination VM. By encapsulating the packet during transmission by the network, NetLord prevents the MAC addresses of virtual machines to be registered by switches on the way. Thus, only the switch addresses are stored in the routing tables.

The encapsulation of traffic at virtual machines was necessary to enable one of the main features of NetLord, which is the use of multiple simultaneous routes between devices. SPAIN [13] is application developed by the same group that takes advantage of all routes known to route traffic more efficiently, optimizing the band available.

NetLord main problem is that it requires change at the virtualization server to enable the encapsulation. Another restriction of implementation is its inability to deal with the direct communication between two customers.

Diverter [7] is an application developed by HP Labs, which has similar objectives to DCPortalsNG, such as allowing virtual networks to share the physical resource and at the the same time ensures the insulation between them. Diverter requires changes in the server virtualization. They developed a module, called VNET, which is installed on each server, and that intercepts all traffic that arrives or leaves the equipment. An important difference between Diverter with respect to the DCPortalsNG is that the first uses an arbitrary assignment of IP for the VMs. Thus, there is no collision between IPs any VM, making the problem simpler.

Some of the authors of one of the early work on SDN [4] presented a new work [12] describing a system, called NVP, to manage networks of virtualized datacenters. The tool makes use of SDN in conjunction with arbitrary network topology information created by each customer to build a

traffic isolation solution between participants. Unlike DCPortalsNG, which works reactively, the NVP is based on the topology information and pre-calculating all flow rules. The controller only communicates with the switches when the topology changes, which requires change to the set of calculated rules. The cost calculation is extremely high because there is an explosion of rules. It is necessary to provide all existing communication possibilities, ignoring the fact that we rarely communicate with all their peers [11]. The article presented some tests. In a network with 3,000 virtualization server and more 60,000 ports, this calculation took up to 1 hour to complete. In NVP, isolation between clients on the physical network is achieved using encapsulation.

3. IMPLEMENTATION

The implementation of DCPortalsNG was divided into three distinct modules. The first one is a driver for ML2, which implements the functions necessary for the API that plugin. This small application receives topology changes that occur in the OpenStack and forwards it to the storage in DCPortalsNG, explained below.

The second module is a storage service that communicates with the driver. This module stores persistently the topology information of virtualized networks and makes them available to the application that will define packet flows. The information stored in this module differ from those staying at the ML2 because they are organized to optimize queries that will occur in operation with the SDN controller.

The third module executes using POX and is responsible for handling the OpenFlow requests. Messages received by the controller are transferred for this application, that responds defining which flows rules should be installed to allow communications between virtual machines. This module is the main application of DCPortalsNG and contains all the routing logic and rewrite packets.

One of the main design decisions in relation to traffic isolation was to choose packet header rewriting despite the inherent simplicity of encapsulation. The clearest advantage of the encapsulation is that it does not limit the types of packages that are supported. On the other hand, it overloads the boarder switches during encapsulation, and there is the risk of fragmentation.

This project choose the packet header rewriting as an isolation method because we believe that it is sufficient to meet the objectives, maintaining an efficient and simple translation rule system.

3.1 Communication to OpenStack

Communication between OpenStack and DCPortalsNG occurs through a driver implemented and installed by the ML2 plugin. Given ML2 the resources, the construction of this driver is simplified. It receives query requests, changes the topology of virtual networks, and passes them to the persistence module.

Since it only implements the appropriate functions for its operation, DCPortalsNG considers that whenever the driver is called, a change in topology of virtual networks of OpenStack occurred. Since ML2 maintains the persistence information sent by Neutron, the only task driver does is to pass the data to DCPortalsNG module, which takes care of storage.

The communication between the driver and the storage application occurs in form of remote procedure calls, using messages like JSON-RPC[1], which is a method for executing functions remotely, using simple messages JSON type [5]. The main reasons for using this mode of communication are its simplicity of integration to the already developed code, ease of debugging, and its robustness since it does not depend on system states.

3.2 Architecture

Figure 1 displays the path of the data within the OpenStack platform and between it and the DCPortalsNG. In the external plane of Figure 1 appears the two responsible entities: OpenStack is on the left and DCPortalsNG is on the right. The intersection between the two is through the driver that runs next to the ML2.

In the figure, the OpenStack ① control, which is composed of several interaction interfaces with customers, communicates with the components that apply the changes, such as Nova ② and Neutron ③. This communication uses a message queue protocol called RabbitMQ[2] which is represented byc̃ircled4.

In the case of network changes, they are sent to the Neutron component. This, since being only an abstraction, forward all requests it receives to the plugin that actually will execute the changes. In our system, this task is the responsibility of ML2 ⑤.

The driver developed in the project is configured within the ML2, allowing this to communicate with DCPortalsNG ⑥ module, in executing on POX controller ⑦. Through this sequence of steps, the modifications that were initially received by Neutron arrive to the SDN controller.

Changes in the structure of the virtual machines are sent to the Nova. Nova, besides taking care of the VMs life cycle, connects to the virtual switches ⑧, which are controlled by the SDN controller ⑦.

When there are changes in the states of switch ports, such as when the interface of a VM is connected, these entities inform the changes to the POX controller using an OpenFlow ⑨ connection previously established. Likewise, when the switches receive packets that do not match any rules of their routing table, these are passed to the controller, using the same path.

Finally, using the topology information sent by the ML2 and ports sent by virtual switches, DCPortalsNG module is capable of deciding which rules should be applied to packets arriving to switches. So when a switch passes a packet that do no know how to handle, the controller compares the packet headers to the information from its database to determine what action should be taken to that flow.

3.3 Interfacing ML2

Whenever there are networking events, such as changes in the network topology within the OpenStack platform, these produce calls to Neutron, that passes on to ML2. Within the plugin, the DCPortalsNG driver is activated whenever selected types of events occur. This selection and further activation occur by the implementation of function with predefined names. These are called by ML2 at the beginning and end of each type of event.

In the ML2 driver of DCPortalsNG, it was implemented only the functions related to topology change, such as addition and removal of virtual machine ports and those related

[1]http://jsonrpc.org
[2]http://www.rabbitmq.com

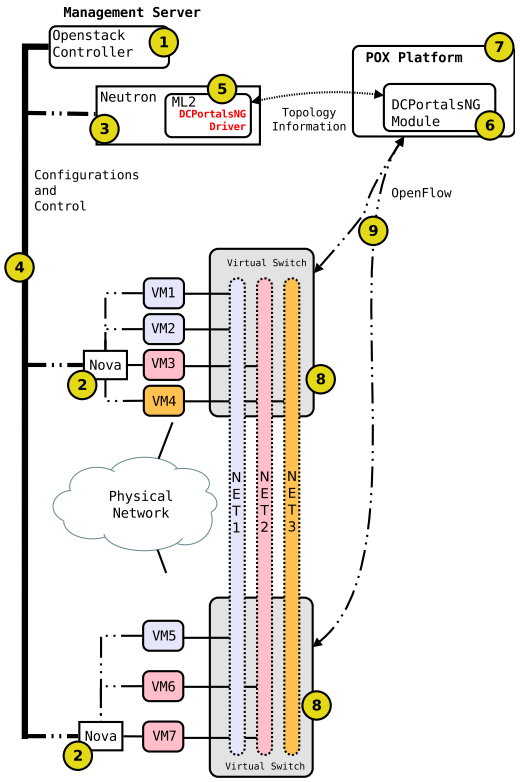


Figure 1: Communication between DCPortalsNG and OpenStack.

to changes in customer networks.

3.4 Data Persistence

When the ML2 driver of DCPortalsNG is activated by the occurrence of a event, it collects all relevant information for that type of event and sends them to the DCPortalsNG persistence module. When forwarding information as a result of events, the driver regularly compares the topology information stored in ML2 to those that are active in persistence module, ensuring synchronization of this information.

Data sent to DCPortalsNG, which are maintained by the persistence module, need high availability and therefore are stored in a database. This, to ensure ACID properties [3], can easily work as a centralized point for queries and modifications of information related to network topology defined in OpenStack.

In order to facilitate the development of DCPortalsNG, we opted for the ORM [1]Object-Relational Mapping programming model to implement the interface to the DBMS (DataBase Management System).

The network topology implemented in DCPortalsNG has 4 different types of objects: switch, network, port and tenant). Figure 2 shows each of these classes along with their properties and functions.

3.5 DCPortalsNG Module Operating in POX controller

Using the switch class presented earlier, DCPortalsNG works closely with the SDN controller across multiple in-

[3]Atomicity, Consistency, Isolation and Durability

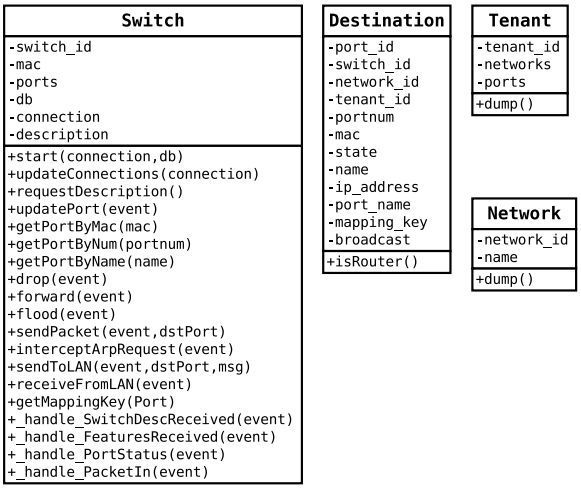


Figure 2: Model of the main objects of DCPortalsNG module internal to the POX controller, along with their functions and important properties.

stances, which are created as new switches establish connections with the controller. Thus, each instantiated object takes care of only one switch, and the coordination of addressing information between them is guaranteed by the use of persistence module that is based on a consistent database.

Following the POX event-driven programming model, the DCPortalsNG module, running in this controller, records some functions that will be called when certain events happen. The starting point of the module operation is through a function that is called when a switch establishes a connection to the controller. At this point, a new instance of the Switch class is created, which goes on to handle and respond to the OpenFlow requests from that switch.

When establishing a connection to the switch, but before treating requests, the instance queries the switch about which virtual machines are connected to it and stores that information in the persistence module. This will act as the point of information exchange between all running instances. After collecting information from the switch, the module starts to respond to all OpenFlow requests originating for it.

The relationship between the various modules that compose the DCPortalsNG are shown in Figure 3. There are 3 main applications, which are the driver in operation within the ML2 plugin, the application that runs along the POX controller to handle OpenFlow events received by it and, finally, the persistence component that stores all the network information that passed on by the two other modules.

In Figure 3 are also presented all the interactions needed to operate the network in the OpenStack platform:

- VMs send data packets to the OvS switch;
- packets that arrive at the switches and do not have associated rules trigger events that are sent to the POX controller through OpenFlow messages;
- a message that reaches the controller from a switch is immediately forwarded to DCPortalsNG module that handles that connection;

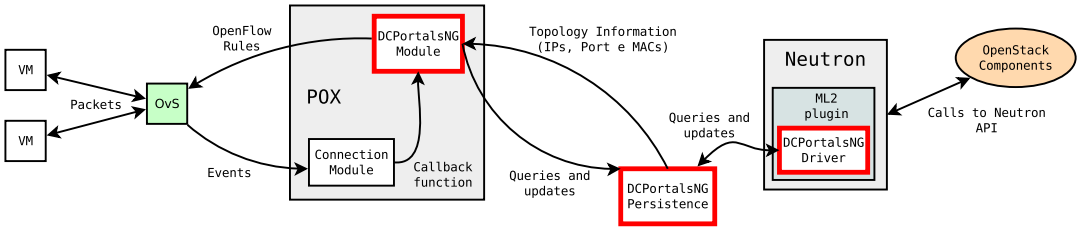


Figure 3: Applications that compose DCPortalsNG and interaction forms between them and the IaaS platform.

- to obtain topology information and to decide the forwarding rule, the internal module to POX queries the persistent module;
- using the retrieval network information, the internal module to POX sends to OvS switch the OpenFlow rules that define how this switch should forward the received packets;
- changes that occur in the topology defined by the OpenStack are stored in DCPortalsNG persistence module after being received by the driver that executes linked to Neutron.

3.6 Mapping destinations

This work proposes to isolate packets of multiple virtual networks that goes through the physical network without the need of special network resource equipments. The isolation is achieved by installing rewritting rules in OvS switches that modify the packets before transmitting them over the physical network.

The DCPortalsNG persistence layer uses objects of type *Destination* to store information related to network interfaces of each of the VMs, which are considered as packet destinations for the application. These data are used by the module that runs on the SDN controller SDN for:

- validate packets that arrive to switches, checking source and destination addresses;
- define forwarding rules by checking the information of which ports belong to which VMs;
- define packet rewriting rules that are transmitted by the physical network, using the map keys associated with each destination.

During the packet rewriting process for transmission over the physical network, it is replaced both the layer 2 and layer 3 addresses. The L2 (MAC) addresses of VMs are replaced by L2 addresses of the OvS switches that are the source and destination virtualization servers. The IP addresses are replaced by mapping keys, which are chosen for each destination arbitrarily by DCPortalsNG. The Table 1 summarizes an example of the information that are used by DCPortalsNG module to construct the rewriting rules.

The only restriction to choose the mapping keys is that they are different for all VMs ports in the same virtualization server. The key allocation occurs on demand, i.e., when there is the need to transmit packets associated to a physical network destination. Furthermore, since the MAC address of all packets going to VMs that are running on the same virtualization server are the same, the physical switches that interconnect servers need to store only one address in their routing tables.

3.7 Forwarding packets

When a virtual machine transmits a packet over the network, the packet arrives first at the OvS switch running in the same VM server, controlled by POX. Since it operates as an OpenFlow switch, it tries to apply the rules of its Flow Table to the packet. If there is an applicable rule, the switch processes the packet according to the rule, forwarding or discarding it. In cases where there are no rules applicable to the received packet, the switch sends it to the POX controller to decide what to be done. Usually, this happens for the first packet of a new flow. The decision is returned in the form of a rule that will be installed in the Flow Table and can be applied to subsequent packets that follow the same flow.

When the POX controller is requested to decide how a switch should handle a packet, it first determines the packet type, its source and destination. After validating this information, DCPortalsNG decides:

- if the packet is an ARP query and DCPortalsNG will answer;
- if the packet should be forwarded to a VM running locally;
- if the packet originated from another virtualization server and needs to be rewritten before local delivery;
- is a packet that needs to be rewritten because it is intended for a VM running on another server;
- if the packet is a broadcast packet and DCPortalsNG will have to converted the packet into multiple unicast packets and forward them to each of the other servers.

Based on the initial analysis described above, DCPortalsNG will take a sequence of actions that will end as responses to the switch, indicating how the switches should handle the packets. Ahead, we will detail every possible sequences of actions.

3.7.1 ARP queries

ARP queries are intercepted by POX controller and answered without broadcasting packets over the network or even arriving at the machine responsible for the requested IP.

The ARP request packet that is sent by the VM reaches the OvS switch. This, for not having a rule on its Flow Table to matche the packet, forwards it to the POX controller. The controller when receiving the message from the switch calls the function `_handle_PacketIn()` from the instance of the object that represents that device.

Function `_handle_PacketIn()` identifies what type of packet was received. If it detects the packet is an ARP request, DCPortalsNG searches the database looking for the associated

Server	OvS MAC	VM	VM MAC	VM IP	Mapping Key	Broadcast
HostA	01:01:00:00:00:01	VM1	02:01:00:00:00:01	192.168.0.1	10.0.0.1	No
HostB	01:02:00:00:00:02	VM2	02:02:00:00:00:01	192.168.0.2	10.0.0.1	No
HostA	01:01:00:00:00:01	VM3	02:01:00:00:00:02	192.168.0.3	10.0.0.2	No
HostB	01:02:00:00:00:02	VM4	02:02:00:00:00:02	192.168.0.4	10.0.0.2	No
HostA	01:01:00:00:00:01	—	—	192.168.0.255	10.0.0.3	Yes
HostB	01:02:00:00:00:02	—	—	192.168.0.255	10.0.0.3	Yes

Table 1: Table of necessary information to rewrite packets

MAC with the requested IP. In addition to the IP address, DCPortalsNG also takes into account the client information and network port by which the packet had arrived at the switch. When nothing is found, the packet is discarded. In case there is a MAC associated with the searched IP, the function constructs an ARP reply packet which is sent as an OpenFlow message to OvS switch. This, in turn, forwards the response to the intial VM that had requested the ARP.

3.7.2 Traffic within the servers

The traffic between VMs running on the same virtualization server is forwarded without modification. When the first packet flow passes by OvS switch, POX controller installs a forwarding rule to treat all subsequent packets.

3.7.3 Traffic between servers

Mapping keys ensures the isolation of customer traffic and conceal the addressing information of the virtualized environment from the the physical network. When one virtual switch must forward packets to VMs running on other servers, rewriting rules replace the addresses of the headers by keys before the packets reach the physical network. Later, when the modified packets arrive at the OvS switch of the server that host the destination VM, this rewrites the packets again, retrieving the original headers, and then forwards them to the final destination.

It is through translation rules that OvS switches are able to rewrite the source and destination addresses packets and enable communication through the physical network. In the example of figure, the virtual machine *VM1*, hosted on the server *HostA*, transmits a packet to destination *VM2*, which runs on the server *HostB*. When the OvS switch of *HostA* receives the first packet from *VM1* to *VM2*, it passes the packet to the POX controller, which in turn passes it to the DCPortalsNG module. The latter, after setting the key mappings, sends a translation rule to the switch. The rule is installed in Flow Table of OvS switch at *HostA*. From that moment, it is able to rewrite the packets destined to the *VM2* without relying anymore on POX controller. When the rewritten packet arrives at the virtual switch of *HostB*, that switch does not know to handle it and a new translation rule must be built from DCPortalsNG to retrieve the original headers.

For the communication example of Figure 6 if the DCPortalsNG had the addressing information previously submitted in Table 1, the transmitted packet between *VM1* and *VM2* would be rewritten as shown in Figure 7. The figure shows the packet headers during important points on the path: exiting the machine *VM1*; after the application had rewritten the rule in the source OvS switch, that replaces the packet address by the mapping keys; the packet received by virtual switch on the destination virtualization server; after recovering the original address with the application of the translation rules for the second *O*vS switch and finally after the packet had being delivered to *VM2* machine.

3.7.4 Broadcast IP Traffic

When a broadcast packet, originated in VM directly connected, reaches a virtual switch, the first action done by that switch is to check if it is a message of type ARP Request. If it is an ARP traffic, it is intercepted by ARP process explained above, which handles the message. If it is an IP broadcast packet IP, it uses a forwarding process that converts the message into unicast messages before sending it to the physical network.

The forwarding process of broadcast packets by the network physical network resembles the approach adopted for unicast packets. The main difference is that the DCPortalsNG needs to create a dummy destination, associated with each OvS switch, to be able to identify such traffic during the rewriting process. Objects of type *Destination* associated with this communication have broadcast field enabled. During the rewriting for transmission, the destination IP packet header is replaced by key mapping allocated to the dummy destination. That allows the packet to travel the physical network as a unicast type packet and it is recognized as a broadcast when it arrives at the destination OvS switch. When receiving the packet, that switch rewrites the headers, replacing the destination IP header to the broadcast network address to which the packet was originally sent and then the switch forwards it to all ports of local VMs that are in that network.

A side effect of using unicast packets in broadcast transmissions through the physical network is the transmission of multiple packets by the physical network, because of the need to send a packet to each virtualization server that has VMs connected to the virtual network where the broadcast occurred. These multiple packages are built by rewriting rules installed in the OvS switch directly connected to the VM that is generating the traffic.

3.7.5 Traffic to external networks besides the OpenStack platform

During communication between IaaS platform and the external environment, such as example, Internet addresses, the rewriting technique previously described can not be used. If it were applied, there would be not enough keys to map all possible destinations, and that would not be effective because of amount of destinations that would need to be stored. To enable the communication in these situations, we adopted the procedure of only replacing.

When the DCPortalsNG detects that the package to be rewritten have external source platform, it builds the rewriting rules only to replace the MAC and IP address of the virtual machine to which the packet is intend. This detection is possible through the knowledge of the topology: when the source packet is a router, it is considered external and the MAC address of that router and the IP originate outside IaaS environment are left unmodified. In the case of packet from the opposite way, leaving a VM to an external address, the rule changes only the address of the VM that originated

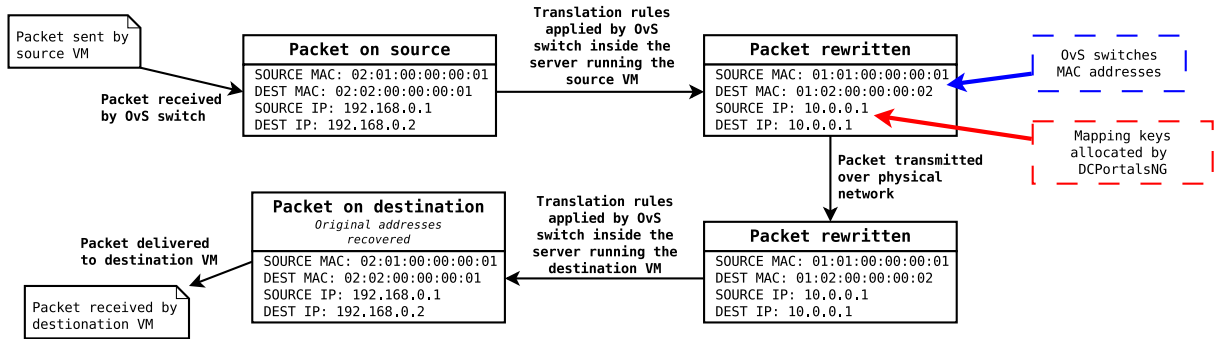


Figure 4: Forwarding packets through the physical network

the packet.

It is important to notice that the partial rewritten occur only when it is necessary to forward packets with source or destination outside the IaaS platform between virtualization servers. This situation occurs when the VM making the communication with the external environment runs on a different server than where OpenStack created the router of the virtualized network.

The fact of not rewriting all addresses for external communication causes the exposure to the physical network of partial addresses used by clients in their virtual networks. However, this visibility is limited, since only router addresses, which are created by the OpenStack platform are visible.

By allowing only the MAC addresses of virtual routers to reach the physical network, it greatly limits the amount of addresses that go to the forwarding table of physical switches. Regarding the isolation of clients, since DCPortalsNG knows the IP and MAC addresses of all virtual machines on the network from a client, it can easily verify the address and avoid spoofing packets by malicious clients.

3.8 Limitations

Since we require to rewrite IP address fields of the packets to ensure isolation between customers and to route them between VMs of different virtualization servers, DCPortalsNG is able to work only with IP and ARP traffic type.

4. SYSTEM EVALUATION

This section describes the environment built to certify DCPortalsNG capabilities and its performance.

4.1 Testing environment

A physical infrastructure simulating a small IaaS provider was built to validate DCPortalsNG's operations. It was constructed to allow us evaluate the main situations that could appear in a real data center.

Figure 8 presents the topology of the testing environment. It shows 3 virtualization servers connected to 2 switches. One of these is used for OpenStack control communications and to access the data center network. This switch is also used for Internet access and communication with the POX controller. The second switch is exclusively used for traffic between virtual machines, simulating the most probable configuration of an IaaS provider. These providers commonly have very big infrastructures, where servers and controllers are spread across the network. It is important to note that both switches do not have any special configurations, and

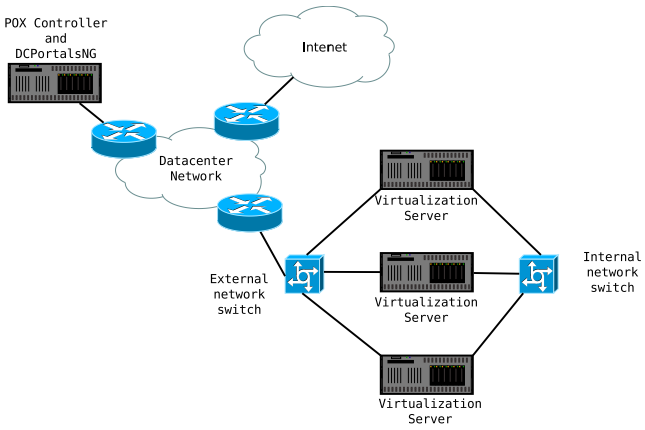


Figure 5: Physical topology of the testing environment.

are working like a dumb Ethernet layer 2 forwarding switch.

Notice that we chose two switches to facilitate the understanding and verification of the correct isolation behavior between the physical and virtualized environment. Since the DCPortalsNG does not use VLANs, they could had been used to construct an environment having the same characteristics with just one switch.

Above the physical structure, we installed a virtual topology composed of VMs from multiple clients, distributed between virtualization servers and connected by private or shared virtual networks. As this topology varied for each test performed, it will be presented with the tests.

The specification of each of the virtualization servers used in the testbed is as follows: 2 Intel Xeon E5440 2.83GHz processors; 16Gbytes RAM; 4 SATA disks 10k-rpm in Raid5; Linux Ubuntu 13.10. For the POX controller, it was used a machine with: Intel Xeon E3-1240 processor; 6-Gbytes of RAM; 1 SAS disk; and Linux Ubuntu 13.10. All machines have 1-Gbps network interfaces and are connected to switches with compatible capacity. The configuration of virtual machines used in all the tests was: 1 processor; 1 Gbyte of RAM; 30 Gbytes disk; Linux Ubuntu 13.10.

In order to validate the operation of DCPortalsNG, the following set of tests was performed: latency tests, isolation tests, band tests,and load tests. We tested 3 different network operation scenarios:

1. DCPortalsNG with POX module;
2. L2 switch from POX, which is offered as reference by

Figure 6: Virtual network topology used in tests of MAC resolution. All logical networks were implemented over the same physical network.

the own controller code, as indicated in the results as POX + L2;

3. OvS switch. It only forwards packets in Ethernet. It had no configured controller nor isolation between client networks.

4.2 Latency

The objective of latency tests is to measure the time before the start of communication between two virtual machines inside the environment. During this phase many actions occur, like MAC address discovery, installation of flow rules into the OvS switches and, finally, packet forwarding.

All stages prior to the beginning of communication were evaluated separately to provide a better understanding of system behavior. We measured the time required for MAC resolution, the time required for installing the flow rules and the time required for forwarding the packets after the flow was established. We also measured the forwarding latency of broadcast packets, since this type of transmission has a special treatment by DCPortalsNG.

4.2.1 MAC address resolution

Because DCPortalsNG intercepts and answers all ARP packets without contacting the destination VM, this communication stage was measured separately. We measured the time interval between a VM sending an *ARP Request* and the *ARP Reply* been received by it. All measurements were done capturing traffic in the network interface of the VM. The tests of this stage were executed when the system was in normal conditions and also when it was under attack.

The MAC resolution following tests were done using the topology presented in Figure 9:

1. Normal situation: VM1 requests MAC address of VM2; VM1 and VM2 are on the same local network NET1;

2. Attack on the local network: VM1 requests MAC address of VM2 while VM3 is flooding the same local network NET1;

3. Attack on remote network: VM1 requests MAC address of VM2 while VM5 is flooding the network NET3;

4. Attack on isolated network: VM1 requests MAC address of VM2 while VM4 is flooding the network NET2.

The difference between tests 3 and 4 is that in the last one there is only one VM in the network. Because DCPortalsNG knows the network topology, it does not forward packets from VM4 when there is no other VM on the same network in other virtualization servers.

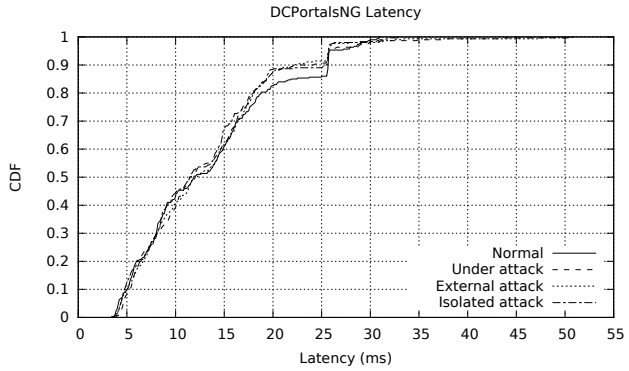


Figure 7: ARP resolution latency using DCPortalsNG.

The following sequence of commands were executed in VM1: "`ip neigh flush all`" and "`ping -c 1 VM2`". These commands clear the ARP table of VM1, forcing it to send and *ARP Request* packet before transmitting ICMP messages.

Instead of using the output of command `ping`, the measurement of the interval between the *ARP Request* transmissions and the *ARP Reply* responses was done by first capturing traffic in the network interface of VM1 and later calculating the time difference between transmissions and receptions. The capture was done with the command "`tcpdump -ni eth0 arp`".

The flood attacks were done by executing the command "`ping -f BCASTLOCAL`" in the attacking VM. In this command, `BCASTLOCAL` is the broadcast network address of the local network. The parameters used with *ping* command forces the forwarding of up to 100 packets per second to every network interface in the same local network of the attacker.

The figures 10, 11 and 12 show the results of ARP resolution using DCPortalsNG, POX controller with the L2 switch application and Open vSwitch without a controller, respectively. Each figure contains 4 lines representing each type of test already described.

As expected and confirmed by the graph of Figure 10, DCPortalsNG had similar response times in every type of test. This was expected because it intercepts the ARP traffic and generates the responses internally, and prevents it suffering from the flood attacks. Furthermore, during the attack in an isolated network, DCPortalsNG avoids that the attack reach the physical network.

In the situation where the POX controller is used with the L2 switch application, the system performance suffers from flood attacks in every case. Because the switch is constantly trying to learn the topology while the packets are transmitted, it ends up forwarding all packets of the attackers. The results in Figure 11 show that the solution has the worst performance during the attacks tested.

Open vSwitch has the best results, as shown in Figure 12. This performance is possible mainly because it executes within the operating system kernel and also because no type of validation is done before packets are forwarded by the virtual switches, when they do not have an OpenFlow controller. On the other hand, all switches and network connections are overloaded during the attacks, because it unconditionaly forwards every packet that is generated during the flood.

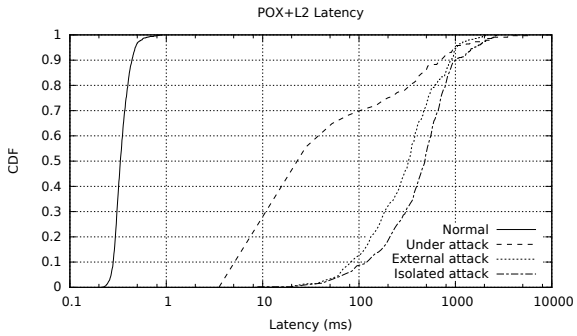


Figure 8: ARP resolution latency using POX+Switch L2.

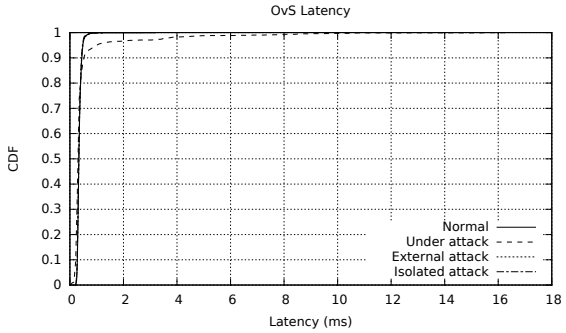


Figure 9: ARP resolution latency using OvS.

4.2.2 *Configuration Latency*

The test where there is communication between different server virtualization is especially necessary because it allows evaluating the cost of packet translation which is held by DCPortalsNG when externally communicating to the virtualization server. As translation rules are installed in both the source virtual switch and the destination server, it is expected that there is some performance loss in relation to the configuration that uses POX with L2 switch application.

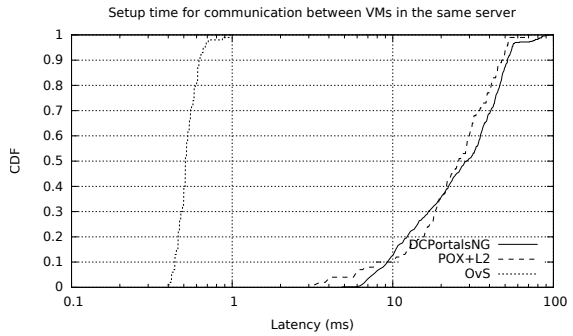
As we see from the results presented in Figure 14 presents the latency results to install OpenFlow rules on swithces. In all situations, the use of a controller, that is working reactively to install the forwarding rules, increases a long delay when compared to OvS switch that uses no controller. Additionally, the second graph shows the additional cost of assembling the translation table that occurs when there is communication between different virtualization servers.

4.2.3 *Latency for established flows*

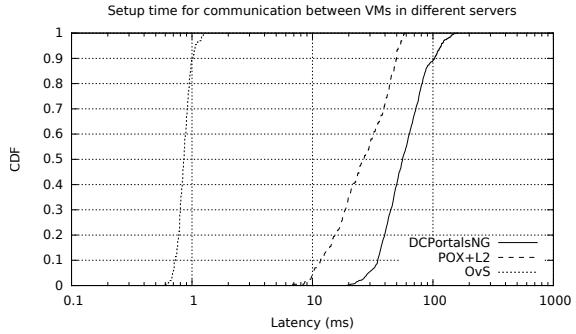
We evaluate the latency of packet transmission in established flows. We measure the latency between a VM in a virtualization server and several different destinations.

For this test, the response times were measured by packets

Figure 10: Topology for the configuration time test



(a)



(b)

Figure 11: Latency for installing OpenFlow rules on switches

Figure 12: Network topology for latency tests.

sent by VM1 to destinations in local and remote points of the virtual network. Figure 15 depicts the topology and destinations that VM1 communicates.

Table 2 illustrates the results. Unlike the previous tests, where there was great variation between the results for each configuration, after the flows are established, this difference becomes minimal.

Dest.	**Conf.**	**Avg**	**Max**	**Min**	**Std**
VM2	DCPortalsNG	0,35	1,19	0,26	0,078
VM2	POX+L2	0,36	0,99	0,25	0,079
VM2	OvS	0,37	0,95	0,27	0,080
VM3	DCPortalsNG	0,32	0,89	0,21	0,070
VM3	POX+L2	0,32	0,74	0,21	0,063
VM3	OvS	0,31	0,70	0,21	0,064

Table 2: Latency results for flows that are established. All results are in milliseconds.

4.2.4 *Broadcast packet latency*

In addition to measuring the various steps involved in the exchange of packets between virtual machines in OpenStack environment, we also measure the transmission delay

Figure 13: Topology for broadcast tests

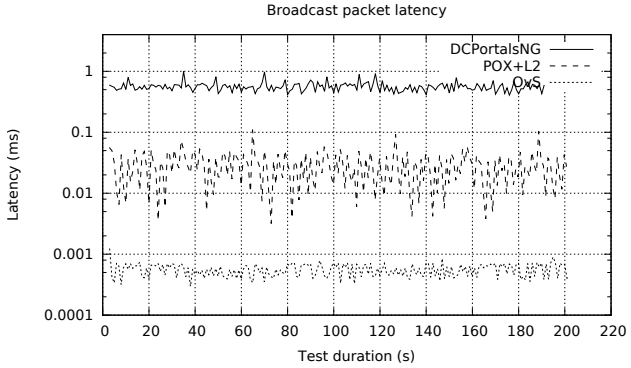


Figure 14: Broadcast Latency.

of broadcast packets. Although we use also ICMP packets for this test, the differential treatment of DCPortalsNG for broadcast traffic is important to evaluate.

Figure 16 presents the used topology, in which 3 VMs from a client share the same virtual network. The latency evaluation was performed with the ping command running on VM1. This command sends ICMP packets destined to the broadcast virtual network address. Because it is disabled by default in Linux kernel, it was necessary to enable the answer to ICMP packets sent to the broadcast address.

Figure 17 depicts the broadcast latency results. One may compare the delays added by including an SDN controller and also by the conversion of broadcast to unicast, used by DCPortalsNG, to the configuration that only uses the OvS switch without controller. The values refers to the average latency of response to ICMP packets *echo request* and the results are presented without any smoothing.

By relying on the package conversion of broadcast to unicast in the transmission over the physical network and to check which servers had virtual machines from clients, the average performance of DCPortalsNG is $0,563\ ms$, while the SDN configuration using POX + L2 is $0,003\ ms$. As the OvS without controller does not check anything, its results were extremely better than the other two, obtaining an average response $0,00056\ ms$. Although slower, it is believed that the use of unicast proves interesting in large environments. There, each client usually has few VMs in relation to the size of the datacenter and executing broadcast as unicast prevents the client to access the entire network. Furthermore, there is little reason to broadcast when the ARP broadcast is removed.

4.3 Available bandwidth

The efficiency of the proposed system in the use of the maximum available bandwidth was measured over TCP transmission tests, which tried to take all of the available capacity. The experiment topology is the same as the one used in the setup time test, which is presented at Figure 13.

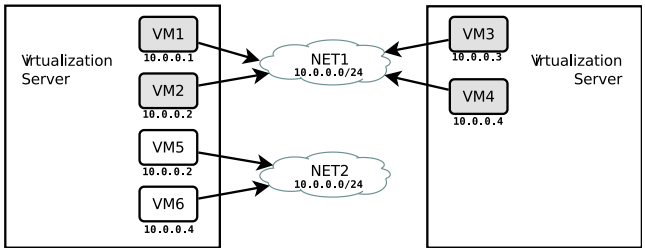


Figure 15: Network Topology during isolation tests.

We use iperf tool to generate synthetic traffic and measure bandwidth between VM1 and the other two virtual machines, VM2 and VM3. The TCP connection was set up to 600 minutes.

Table 3 shows the occupation of the network in each of the evaluated situations. The results demonstrate that the use of an SDN controller does not influence the results. It is important to notice that in communication tests within the same virtualization server, the maximum communication capacity is limited only in software, by OvS. Thus, the results in these cases can reach values of up maximum expected capacity for connections, which are 1 Gbps.

VMs	Conf.	Avg	Max.	Min.	Std
1–3	DCPortalsNG	0,94	1,05	0,84	0,01
1–3	POX+Switch L2	0,94	1,14	0,76	0,03
1–3	OvS	0,94	0,95	0,92	0,00
1–2	DCPortalsNG	0,94	0,95	0,63	0,02
1–2	POX+Switch L2	0,93	0,94	0,74	0,02
1–2	OvS	0,93	0,94	0,93	0,00

Table 3: Available bandwidth test results. Results are in gigabits.

4.4 Isolation between virtual networks

The test for validating the isolation between virtual networks was carried out by sending ICMP packets from multiple virtual machines of different customers using the same IP address block. The topology presented in Figure 18 shows the used environment, which contains a total of six virtual machines distributed between two clients. Each client uses a different private network but all allocated IPs are in the block`10.0.0.0/24`. Since the networks are independent, it is expected that even using the same IP, a client packet do not reach the VMs in the other network.

The test executes the command "`fping -g 10.0.0.0/24`" from each one of the existing VMs in the environment. The detection for received packets is performed by tcpdump, which capture all traffic of type ICMP arriving at the network interface.

In each row of the Table 4, are marked the virtual machines that received the echo request packet sent by the VM indicated in the beginning of the line. As we can see, the DCPortalsNG allowed the packets sent by the VMs to reach only the machines that belong to the same client and were in the same network. Table 5 shows the results for the the same test using the POX + L2 configuration. We also performed the same test with OvS switches and the results were the same as POX + L2.

The results, shown in Table 4, prove the isolation between

virtual networks for each client. With the other two configurations (POX + L2, OvS switch), there is no isolation between networks and all VMs receive all packets, which is exactly the behavior we want to avoid.

Sent by \ Received by	VM1	VM2	VM3	VM4	VM5	VM6
VM1	–	√	√	√		
VM2	√	–	√	√		
VM3	√	√	–	√		
VM4	√	√	√	–		
VM5					–	√
VM6					√	–

Table 4: Results of DCPortalsNG isolation tests

Sent by \ Received by	VM1	VM2	VM3	VM4	VM5	VM6
VM1	–	√	√	√	√	√
VM2	√	–	√	√	√	√
VM3	√	√	–	√	√	√
VM4	√	√	√	–	√	√
VM5	√	√	√	√	–	√
VM6	√	√	√	√	√	–

Table 5: Results of POX+L2 and OvS isolation tests

To demonstrate the importance of isolation, it was performed a throughput load test. While VM1 and VM3 machines performed a TCP transmission, VM6 machine executed a flood attack on broadcast address to its network.

Table 6 presents the achieved throughput for each configuration. Notice that since DCPortalsNG isolates the flood attack within a virtualization server, DCPortalsNG can get a better performance than the other two configurations.

Config.	**Avg Throughput**	**Std**
DCPortalsNG	825 Mbps	2,1
POX+Switch L2	170 Mbps	5,3
OvS	295 Mbps	3,2

Table 6: Results of throughput tests under broadcast attack

4.5 Load Tests

The last set of tests evaluates the system capacity, in particular, in how the POX controller deals with the volume of packets that require new flow rules. Figure 19 shows the topology used for this test, in which four virtual machines are distributed over two virtualization servers and connected to the same virtual network.

The system capacity was evaluated by several metrics, which were collected simultaneously: (*i*) CPU utilization of the SDN controller process - when used; (*ii*) CPU utilization of the process responsible for OvS switch on the virtualization server VM4; (iii) latency in the transmission of ICMP packets between VM1 and VM2; (iv) the capacity to use the maximum available bandwidth between virtual machines VM2 and VM3 through the command iperf. During the tests, the VM4 machine acted as a flow generator, sending increasingly bursts packets to random destinations,

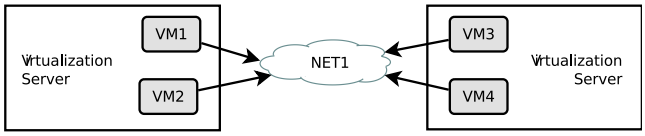


Figure 16: Network topology used for load tests.

forcing the OvS switch to send the new flows to the POX controller. The CPU measurement process of the OVS was performed on the virtualization server VM4 because this machine is the one that needs to address the bursts of packets sent by VM4.

The bursts of packets were spaced periods of 60 seconds to allow the system back to the initial state before another test was initiated.

We measure available bandwidth with iperf command setting a connection between VM2 and VM3. We measure latency with fping every 2 seconds. This time is greater than the time out of flow table rules. Sending packages with this frequency causes all of them to query the controller for the forwarding definition rule, thus allowing us to do measurement about performance loss of the controller response time. We measure CPU load and memory with ps command.

Finally, sending random bursty traffic is done with the tool *packit* [4]. It is able to inject packages of different types and characteristics, including packet size, amount sent per second and randomness of the addresses and ports of both source and destination.

To allow a better visualization of all the information collected and facilitate understanding of correlations involved, the results for that set of tests are presented in a graph that group all measurements in the same image. Figure 20 is composed of over 3 graphs, each corresponding to one or more data types. In the upper graph is presented the CPU usage for each application and the size of the bursts packets. The middle graph shows the achieved bandwidth. Finally, the bottom graph shows the latency when creating new flows through the test with ICMP packets. The X-axis of all graphs refers to the same interval time.

The measurement results with the POX controller with the DCPortalsNG are shown in Figure 20. They show that the developed system begins to suffer performance loss when the amount of flows is about 200 connections per second. Although, it is only capable of handling well a low amount of new connections we must remember that the DCPortalsNG performs several checks before deciding which flow rule should be installed in OvS switch.

Since DCPortalsNG depends on the topology implemented in OpenStack and also needs to access the database to query it, the DCPortalsNG undergoes an overhead higher than other network configuration to handle OpenFlow messages. Despite not having been developed with high performance as its main objective, given POX have been implemented in the *Python* language, we believe that it can be improved by adjusting the used components or rewritten of the functions that require higher CPU usage.

5. CONCLUSIONS

In this work, we presented DCPortalsNG, a system designed to provide traffic isolation for virtualized networks in

[4] http://packetfactory.openwall.net/projects/packit/

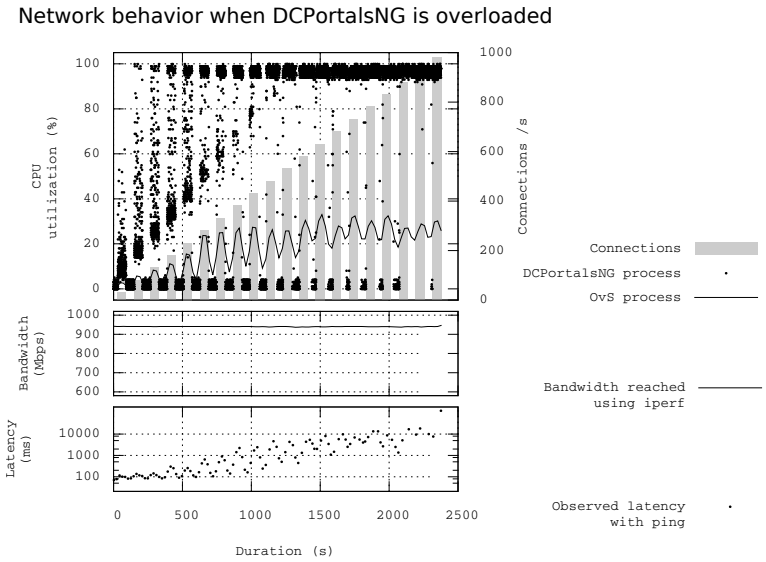


Figure 17: Load test results for DCPortalsNG.

data center environments. Our approach rewrite packages to prevent virtualized customer traffic to affect the physical network. The biggest advantage of our approach is that it does not require any additional feature from the forwarding network elements in the physical environment. Our approach allows VLANs to be used for other purposes by the data center administrators such as the separation of the administrative traffic from the customer traffic.

DCPortalsNG works together with OpenStack, a widely used IaaS platform, and makes the SDN controller responsible for orchestrating the use of network resources.

The system architecture and implementation details, together with Test results confirm the isolation.

Unlike solutions that make use of encapsulation to isolate multi-tenant traffic, DCPortalsNG isolates traffic by rewriting packet header. Virtual OpenFlow switches, which operate between the physical and the virtual environment border, handle the rewriting.

We validated our system in a physical testbed. The results shows that we are able to isolate traffic. We also quantified the system performance under overloads. Finally, we show that the system can be effective in protecting the network from DoS attacks within the data center network. In addition, the results show that DCPortalsNG get similar performance to other solutions after communication between network nodes is established.

In the future, we will evaluate the feasibility of generating proactive rules instead of reactively. Such behavior (proactive) has been advocated by SDN creators as preferable whenever possible. Another idea is to integrate DCPortalsNG to a system that ensure quality band within the data center environment.

6. REFERENCES

[1] D. Barry and T. Stanienda. Solving the java object storage problem. *Computer*, 31(11):33–40, Nov. 1998.

[2] S. Cabuk, C. I. Dalton, A. Edwards, and A. Fischer. A comparative study on secure network virtualization. Technical Report HPL-2008-57, HP Laboratories, 2008.

[3] S. Cabuk, C. I. Dalton, H. Ramasamy, and M. Schunter. Towards automated provisioning of secure virtualized networks. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 235–245. ACM, 2007.

[4] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. *ACM SIGCOMM Computer Communication Review*, 37(4):1–12, 2007.

[5] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). Technical report, The Internet Engineering Task Force, 2006. `http://www.ietf.org/rfc/rfc4627.txt`. Acessado em novembro de 2013.

[6] D. D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). Technical report, The Internet Engineering Task Force, 2000. `http://tools.ietf.org/html/rfc2784.html`. Acessado em fevereiro de 2014.

[7] A. Edwards, A. Fischer, and A. Lain. Diverter: A new approach to networking within virtualized infrastructures. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, pages 103–110, New York, NY, USA, 2009. ACM.

[8] B. Heller, D. Erickson, N. McKeown, R. Griffith, I. Ganichev, S. Whyte, K. Zarifis, D. Moon, S. Shenker, and S. Stuart. Ripcord: a modular platform for data center networking. *ACM SIGCOMM Computer Communication Review*, 40(4):457–458, 2010.

[9] T. Jeffree. IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks. IEEE 802.1Q – disponível em `http://www.ieee802.org/1/pages/802.1ad.html`,

2005.

[10] T. Jeffree. IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks—Amendment 4: Provider Bridges. IEEE 802.1ad – disponível em `http://www.ieee802.org/1/pages/802.1Q-2005.html`, 2006.

[11] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 202–208, New York, NY, USA, 2009. ACM.

[12] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang. Network virtualization in multi-tenant datacenters. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 203–216, Berkeley, CA, USA, 2014. USENIX Association.

[13] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 18–18, Berkeley, CA, USA, 2010. USENIX Association.

[14] 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*, SIGCOMM '11, pages 62–73, New York, NY, USA, 2011. ACM.

[15] OpenStack. Neutron ML2. `https://wiki.openstack.org/wiki/Neutron/ML2`, 2013. Acessado em setembro de 2013.

[16] OpenStack. *OpenStack Networking Administration Guide*. OpenStack Foundation, 4 2013.

[17] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby. Virtual switching in an era of advanced edges. In *Proceedings of the 2nd Workshop on Data Center – Converged and Virtual Ethernet Switching (DC-CAVES)*, 9 2010.

[18] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. Technical report, The Internet Engineering Task Force, 2001. `https://www.ietf.org/rfc/rfc3031.txt`. Acessado em fevereiro de 2014.

[19] T. Sridhar, L. Kreeger, D. Dutt, C. Wright, M. Bursell, M. Mahalingam, P. Agarwal, and K. Duda. VxLAN: A framework for overlaying virtualized layer 2 networks over layer 3 networks. Technical report, The Internet Engineering Task Force, 2014. `http://tools.ietf.org/html/draft-mahalingam-dutt-dcops-vxlan-07`. Acessado em janeiro de 2014.

[20] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying nox to the datacenter. In *Proceedings of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.