DCPortalsNg: Efficient Isolation of Tenant Networks in Virtualized Datacenters

Heitor M. B. Moraes, Rogério V. Nunes, and Dorgival Guedes
Department of Computer Science
Universidade Federal de Minas Gerais
Belo Horizonte, MG – Brazil
Email: {motta,rogervn,dorgival}@dcc.ufmg.br

***Abstract*—Multi-tenant datacenters have become an important scenario in this age of Cloud Computing. One important element for their effective deployment is the isolation of traffic from each tenant within the datacenter. In this work, we present *DCPortalsNg*, a solution based on the Software-Defined Network (SDN) approach, which provide effective and scalable isolation for virtualized datacenters. By adopting a per-tenant virtual network description, we can use an SDN controller to rewrite packets that flow through the physical network. That way, we can easily control which virtual machines they can reach. In our implementation, we leverage OpenStack Neutron's network representation to achieve a simple and extensible solution. Experiments show good results with little overhead, even preventing DoS attacks between tenants.**

Keywords*—*Software Defined Networks; virtual networks; OpenFlow.

I. Introduction

With the large adoption of Cloud-based solutions, multi-tenant, virtualized datacenters have become an important deployment scenario. In them, each user (the tenant) uses the datacenter hardware to host a set of virtual machines, configured to provide a specific service to his clients. One important element in that architecture is the proper isolation of traffic between tenants. That is necessary to guarantee the privacy and safety of each tenant's data, as well as to avoid unexpected traffic (malicious or not) to hurt the performance of any application in the datacenter. While machine virtualization provides good CPU, memory and storage isolation, there is still a need for better network virtualization solutions, specially when scalability and easy management are also expected [1].

Some datacenters rely on the use of VLANs to isolate each tenant's traffic. Although relatively simple, VLANs are limited by the protocol, and in some cases that may hinder scalability [2]. On such solutions, there is still the problem of handling the addresses of the large number of VMs in a single datacenter network. In some cases, each physical machine can host on the order of hundreds of VMs, each with its own layer 2 (MAC) address. Forwarding tables in most Ethernet switches have limited space, and performance can drop significantly if they cannot hold all addresses observed. Solutions based on protocol layering (tunneling) can reduce the need for switches in the network to learn all addresses, but they have performance and management limitations. A good review of these techniques is the work of Cabuk *et al.* [2].

With the advent of Software Defined Networks (SDN), it has become possible to program the network, so that new functionalities and behaviors can be added to switches in the network. Some new protocols and network architectures are being proposed to address those issues, but they require new hardware to operate [3], [4]. In most cases, upgrading all the network hardware is not an option, and scalable, cost-effective, easy-to-manage solutions are still missing. In a previous work [5], we proposed a software-only solution using an SDN controller and the virtual switches at the edge of the (virtualized) datacenter network. Although effective, that solution had some major limitations, primarily in the way tenants could represent their virtual networks and in the way it achieved isolation, which limited the tenants hability to use any IP addresses they chose for their networks (specially restricted addresses).

In this paper, we describe *DCPortalsNg*, which addresses those issues without requiring new hardware. We make use of the new OpenStack Neutron component (http://www.openstack.org/) to handle the tenant networks and therefore create a more detailed and flexible (yet simpler) representation. With that representation, a better packet rewriting scheme can be applied that, at the same time, gives tenants more flexibility to define their address schemes, and simplifies the processing of packets as they cross the network. By doing that, we hide real traffic origins and destinations from the core of the network (hardware), also hiding traffic from each virtual network from any VMs not belonging to the same tenant.

The abstraction of the SDN network hypervisor provides a logically centralized location where network configuration and control can be performed easily, while maintaining the scalability of the solution. There are important benefits to this approach, like *(i)* a reduced demand for the conventional switches' forwarding memory, since VM addresses are hidden from core switches, *(ii)* the creation of isolated virtual networks for each tenant, guaranteeing that one tenant's traffic will never reach VMs of others, *(iii)* integration of VM and virtual network configuration and management, by integrating the network hypervisor with OpenStack, *(iv)* it leaves VLAN tags free to be used for other purposes, such as implementing VLAN-based multi-path routing, for example [6].

With that in mind, the remainder of this paper is organized as follows: Section II describes the architecture and operation

 ISBN: 978-1-61208-318-6

of the system, while its behavior is evaluated in Section III. After that, Section IV puts *DCPortalsNg* in context, describing related work, and Section V provides some concluding remarks, as well as some observations about future work.

II. Implementation

DCPortalsNg was implemented as a network hypervisor module built on top of the POX SDN controller (http://www.noxrepo.org/pox/about-pox/). It interfaces with *OpenStack* through a Neutron plugin, which provides the information it needs about virtual machines and their virtual networks, such as tenant identification, other VMs in a given network and, specially, VM location. With that information, it builds OpenFlow messages to tell the Open vSwitches how to handle packet flows from/to a given VM. OpenStack controls the hypervisor in each host, which configures its Open vSwitch accordingly. Figure 1 illustrates relations between the modules.

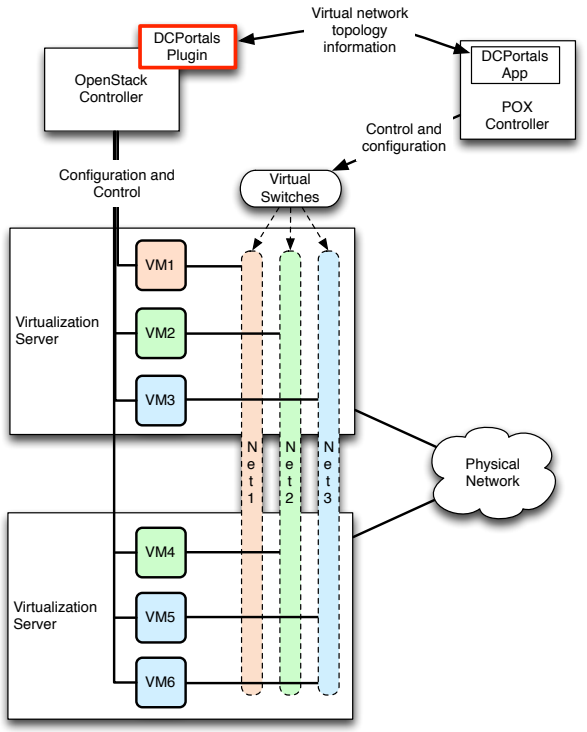

Fig. 1. Architecture of the *DCPortalsNg*.

The system administrator uses the OpenStack API to describe the virtual network for each tenant and to start each virtual machine which belongs to it. OpenStack selects the physical machine that will host the VM and sends the information it needs to run. Through the Neutron plugin, it informs *DCPortalsNg* about the network topology and the VM connections. The hypervisor connects that VM to the Open vSwitch of that physical host. POX can, then, assign a unique ID for each VM located in a certain host (VM IDs are unique for the whole network, but are kept organized by host machine). That information is cached in memory for quick access, when needed.

When the new VM sends its first packet through the network, the Open vSwitch identifies a new flow and uses the OpenFlow protocol to inform POX. It then notifies *DCPortalsNg*, which handles the packet, accessing its data structures to recover the information about the source and destination VMs. If the two machines are not in the same network the flow is not allowed and the packet is dropped. Otherwise, the system commands POX to send another OpenFlow message back to the appropriate Open vSwitch, telling it how to handle all future packets between those two VMs. In *DCPortalsNg*, a flow is identified by origin/destination MAC addresses only, so it will apply to all traffic between the same VMs. The sections that follow provide more details about this process and each of the main aspects of the system.

A. The virtual network abstraction

DCPortalsNg takes advantage of OpenStack Neutron to derive its information about each tenant's virtual network topology. Neutron manages three kinds of entities for each tenant: the network, which can be seen as a switch connecting all the VMs from that tenant, the subnet, which defines the address range for the network, and the ports, that represent the connection of each VM to the network. With that representation, each tenant determines which machines are allowed to communicate to each other (those connected to the same networks) or not. That abstraction guarantees at least the security of a local network, without forcing the tenant to worry about the physical, shared, infrastructure.

Our system builds a set of directories to hold that information: one mapping networks to tenants and *vice-versa*, one to hold the info about ports (VMs) associated with each network, and finally one to map VMs to physical hosts. In all cases, VMs are represented by unique identifiers created by the system as addresses in a 10.0.0.0/8 address space (the same principle could be used with IPv6, which is supported by OpenFlow since version 1.3). Although represented as an IP address, that identifier has no direct relation to the VMs' real IP addresses. With that representation, given a VM identifier, *DCPortalsNg* can recover its network, MAC and IP addresses, physical host and even the Open vSwitch port to which it is connected. That information will be used during the decision process needed to route packets.

B. Packet rewriting for network isolation

As previously discussed, using packet rewriting to implement network isolation has two major benefits: it guarantees that traffic from a tenant will be out of reach for others, and it reduces the pressure on physical network devices to handle MAC addresses for all virtual machines in the datacenter. To achieve that, we rewrite the MAC addresses in all packets that traverse an Open vSwitch at the edge of the network to remove the VM information.

For now, we can assume that *DCPortalsNg* has already identified the associated flow and programmed the edge switches accordingly to rewrite the packet before forwarding it. The process described next is illustrated by Figure 2. In it, virtual

 ISBN: 978-1-61208-318-6

machine *vm1*, operating in physical host machine *host1*, sends a packet to another virtual machine *vm3* in the same virtual network, but physically located in physical host machine *host2*. The original packet sent by *vm1* that will reach the virtual switch at *host1* will contain the MAC addresses of *vm1* and *vm3*, and the IP addresses of both.

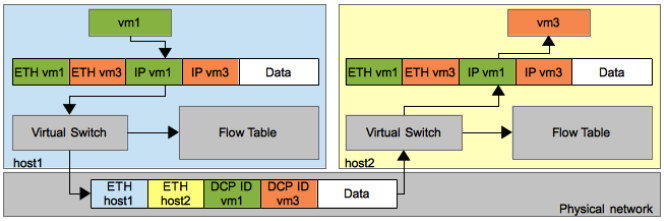

Fig. 2. Address rewriting in *DCPortalsNg*

The Open vSwitch at *host1*, then, will replace *vm1*'s and *vm3*'s MAC addresses in the Ethernet packet header with the MAC addresses of *host1* and *host2*. At the same time, it will replace their IP addresses by their *DCPortalsNg* internal identifiers. Routing at the core Ethernet network will be done in terms of the physical machines' MAC addresses. When the packet reaches *host2*, it will traverse its virtual switch; at that moment, an OpenFlow rule already set in place by *DCPortalsNg*, based on the IDs of the two VMs, will write back the appropriate MAC and IP addresses corresponding to *vm1* and *vm3* in the header. That is the packet that will be delivered to *vm3* at that point. Notice that the MAC addresses of *vm1* and *vm3* never crossed the network; nevertheless, they will only reach their destinations if the system can verify their connectivity to a same virtual network.

C. ARP messages

The MAC rewrite technique just discussed replaces the Ethernet addresses of virtual machines in packets that were already built with those addresses. However, for the VMs to build those packets in the first place, they must learn the MAC address of the destination. In a traditional network, that would be achieved by a broadcast message using the Address Resolution Protocol (ARP). The protocol is composed basically by two kinds of messages: ARP Request and ARP Reply. The first one is sent to the network broadcast address when we need to learn the MAC address associated with a certain IP address. The second is the response, sent by the target machine, to inform the sender of the query about its MAC address.

For the virtual network isolation to work, it is not acceptable that broadcast messages travel the network carrying virtual machine MAC addresses. *DCPortalsNg* fixes this by intercepting all ARP communication and handling it directly. Since it has access to the OpenStack database, it knows how to answer to any ARP query in the network. All it has to do in this case is to build an ARP Reply message with the right information and send it directly back to the appropriate host. Since queries are intercepted at the virtual switch close to the sender, ARP messages never cross the core network.

D. Inter-networking

In modern datacenters, there are cases where two tenants may allow their applications to communicate with one another (based on some mutual agreement). There are also many cases where a tenant's machines must be accessible by clients from the Internet. To achieve that, networks described by the tenants may include mentions to special routers.

The details of how this connection is defined depend heavily on each datacenter structure and service policy. It might be offered only through the definition of a second interface in one of the virtual machines, which would be the only one visible externally, while that host would be responsible to route messages between the internal, virtual network and the outside network.

Although these might be implemented as actual multi-homed VMs, the SDN approach allows us to simplify that, avoiding the need for extra virtual machines: *DCPortalsNg* can simply add rules to directly rewrite packets from the origin network to the destination network, by using special MAC addresses to identify the (abstract) routers.

E. Broadcasts

Although most broadcasts in local networks are ARP-related and, therefore, eliminated by *DCPortalsNg*, we must still consider how other broadcasts are to be handled. When a group of VMs is configured in a virtual network, we expect packets sent to that network's broadcast address to be delivered to all machines in that virtual network, and only to them. However, in a complex, shared environment like current datacenters, that is not the case, since packets would be delivered to all machines connected to the physical Ethernet network.

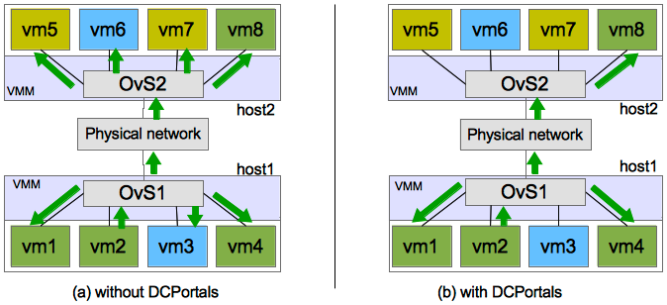

Fig. 3. Difference of treatment of a broadcast

Figure 3 shows the difference between what happens in that case with and without *DCPortalsNg* in the network. In the figure, virtual machines are separated in different virtual networks according to their colors and *vm2* sends a broadcast message. Ideally, that message should be delivered only to the other virtual machines in the same network, *vm1*, *vm4*, and *vm8*. Without *DCPortalsNg*, however, in the conventional system, all machines, no matter to what virtual network they belong, would receive the packet.

 ISBN: 978-1-61208-318-6

To achieve the desired effect, first the broadcast packet is inspected by the virtual switch at the physical machine where the sender VM runs. It is then delivered to the ports of the local switch that are connected to other VMs in the same network, and then *DCPortalsNg* must make sure it is received by all other VMs of that tenant located in other hosts.

The transmission to reach other hosts may be done either by an Ethernet broadcast or by packets addressed to the physical interface of each host holding VMs from that network. One solution has lower latency, while the other will avoid broadcasts flooding a large network.

In any case, when the packet leaves the host machine of the sender, the process of MAC rewriting works as before, to remove the MAC address of the sender, but keeps the Ethernet broadcast address as destination (in the first approach). When the broadcast message reaches the virtual switches at the destination physical machines, in both cases, the controller restores the original headers and also searches for all local virtual machines that belong to the network of the sender, delivering the message to each of them, and only to them. In our example, the controller would program the switch OvS2 to deliver the packet only through the port connected to vm8. By doing that, the message continues to have the effect of a broadcast, but it will only be delivered to machines in the same virtual network of the sender, guaranteeing no other machine will have access to the packet.

For the evaluation tests described later, *DCPortalsNg* maintained the message as a broadcast in the physical network, since that would stress the network further.

III. Evaluation

To perform the validation experiments to confirm the proper operation of the system, we used three machines, each with two network interfaces, connected to two different switches. One of the resulting networks was used for management traffic (OpenStack commands, OpenFlow), and the other was used for the communication between the VMs (operational network). Such configuration is very common in commercial datacenters [1].

The management network uses IP addresses in the range 10.0.254/24. Virtual machines were configured in two separate virtual networks, each spanning two physical hosts. (A single address range was used for all VMs, to stress the need for traffic isolation: being configured with IP addresses in the same range, unless some external isolation is active, traffic from a host can reach all the other hosts.) Two virtual networks were created and machines were distributed according to the configuration described in Table I.

Host machines were configured with the Ubuntu Operating System, version 11.10, and packages *OpenStack*, *Xen*, and *Open vSwitch* from official repositories. The virtual machines were configured with one of OpenStack's standard images, running Ubuntu 10.10.

The first experiment was a simple isolation test using ping to the network broadcast address and the network latency. The second one evaluated the interference of the system on communication latency, also using ping. Finally, the last one tested the system under a denial-of-service condition.

TABLE I
Distribution of virtual machines and their networks among the physical machines for the evaluation tests.

Virtual Machine	IP	Virtual Net	Host
vm2	10.0.20.2	lan1	host1
vm3	10.0.20.3		
vm4	10.0.20.4		host2
vm5	10.0.20.5	lan2	
vm6	10.0.20.6		

A. Isolation and latency overhead

The confirmation of isolation consisted in using ping to send an ICMP Request messages from one of the virtual machines to the network broadcast address (the operating system of the VMs was configured to enable ICMP replies to broadcast requests). The expected behavior for this use of ping is that all machines in the same network as the sender should reply to the sender. When using *DCPortalsNg*, even with all machines using the same IP address range, with the same broadcast address, and sharing the same infrastructure, only virtual machines in the same virtual network as the sender should receive the request and, therefore, reply to it. That was the observed behavior in each case, confirming the proper isolation (program output removed due to space limitations).

We also collected packet traces at the interfaces of the virtual machines (therefore, inside their virtual networks) and at the network interface of the physical hosts (at the point where a packet enters the network core, past the virtual edge switches). By inspecting those traces, we confirmed that packet rewriting occurred as expected.

To verify general latency overheads, we also used ping, now between two VMs in a same virtual network. We discarded the first packet, to discard the ARP and OpenFlow setup overheads (discussed next). We repeated the pings 10,000 times for each scenario and computed a 99% confidence interval for the results, which showed that *DCPortalsNg* overhead in this case was approximately 1% in the worst case.

B. Network setup overhead

It is important to quantify the impact *DCPortalsNg* may have on network latency for a working flow, considering the configuration setup activity. For this experiment, we again used ping messages. We considered three scenarios: first, we configured all Open vSwitch instances to operate as standard Ethernet switches — the default setup, which is used in traditional network configurations. That gives us our baseline case. Next, we configured the Open vSwitches as OpenFlow switches, but used a streamlined network hypervisor that just emulated the operation of an Ethernet learning switch. That would show us the overhead of just having OpenFlow active, without adding the costs of our system. Finally, we considered a complete *DCPortalsNg* installation, where all those costs were included.

 ISBN: 978-1-61208-318-6

First, we evaluated the overhead for the first packet of a flow, when the sequence of actions differs the most between a traditional network and an SDN. As discussed earlier, when that happens in an SDN, the packet is forwarded to the network hypervisor, which must decide what will be done with that flow and send a command back to the switch. In *DCPortalsNg*, that will include executing queries to OpenStack to retrieve information for each VM involved. That may also include an ARP query. We considered both the cases when that query is necessary and when the ARP table at the machines already contains the MAC addresses for the endpoints involved in the communication. When measuring ARP overhead, we isolated its processing by POX by pre-programming the switches forwarding tables, so the network hypervisor would be contacted only to process the ARP request, not for the actual flow. On the other hand, when measuring the flow setup overhead, we pre-programmed the ARP tables, so that no ARP queries were issued. As a baseline, we also measured times for a standard system, where Open vSwitches were configured as standard Ethernet switches, with no OpenFlow. For each scenario, we ran 30 pings and recorded the round-trip times observed. We also computed the statistical difference between each pair of scenarios. Table II shows the average results, with errors for a confidence interval of 99%.

TABLE II
SETUP OVERHEAD FOR EACH SCENARIO, FOR BOTH ISSUING AN ARP QUERY AND INSTALLING A FORWARDING RULE AT THE EDGE SWITCHES. THE STANDARD SWITCH CASE IS SHOWN AS A BASELINE.

Scenario	ARP overhead (ms)	Flow setup (ms)
Standard switch	7.45 +/– 0.1	0.20 +/– 0.03
POX L2 switch	10.72 +/– 2.67	29.83 +/– 8.36
DCPortalsNg	15.31 +/– 3.57	47.05 +/– 8.96

We see that ARP costs vary less than those of flow setup. That is due to the fact that in a standard scenario, ARP requires a broadcast that will reach the destination and a message back; for the POX L2 switch, there is still a network hypervisor involved, but all it does is to return the packets to the switch for delivery as it would be done in the standard switch; finally, for *DCPortalsNg*, an ARP query is transformed into a message to the network hypervisor, which searches an internal table for proper info, and a reply is sent directly to the original sender (there is no contact to the destination machine).

Flow setup costs, however, have a higher variance. There is basically no setup cost for a standard switch; the time shown, 0.20 ms, is just the ping round-trip time through the network. The POX L2 switch must contact the network hypervisor, which will reply by installing a flow based on the addresses it learns during the process, so we can consider that the OpenFlow processing overhead. *DCPortalsNg* adds to that the cost of querying its dictionaries to identify the endpoints and set up the forwarding table. Although there is a significant setup overhead in this case, it is important to remember that it only takes place at the beginning of the communication between two VMs when a flow is set. Besides that, the overhead is less than 50 ms, which would not trigger retransmissions in a TCP connection.

C. Denial of service attack protection

One common motivation for virtual network isolation is the threat that a tenant may start a denial of service attack targeted at another tenant's machines. In a datacenter environment where there is no such isolation, a UDP flow created from an attacking machine to the target network may drain network bandwidth to a point were the attacked system cease to function. One similar attack happened to the BitBucket service, while using Amazon EC2 infrastructure [7] (although, in that case, the UDP traffic came from outside the datacenter).

Such a problem should not happen if the tenants' virtual networks were properly isolated from each other and from the outside. To verify that, we created a UDP flow attack to another virtual network. To make things worse, the UDP flow was created with the broadcast address of the target network as destination. Figure 4 shows the experiment setup in this case.

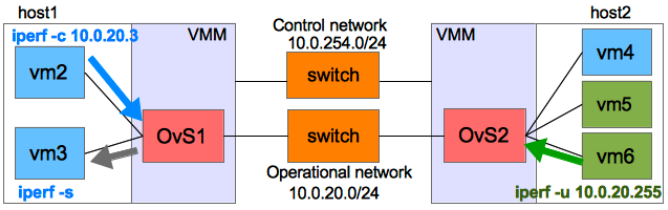

Fig. 4. Denial-of-service experiment. VM6 launches a UDP flood to the network broadcast address, while VM2 and VM3 (in another virtual network) set up a TCP flow.

Virtual machines vm2 and vm3, located at physical host 1, set up a TCP transfer between them. At the same time, vm5, the attacker, initiated a UDP flow addressed to the other virtual network's broadcast address. The attack to the broadcast address is a worst case scenario, since it would be processed by both vm2 and vm3, if delivered. If the attack was addressed to any machine's IP address only, the effect with no control might be slightly less damaging, but the behavior with *DCPortalsNg* would be the same. We used iperf to create the two flows, and measured the effective throughput of the TCP connection between vm2 and vm3. We limited the bandwidth of each virtual machine to 1 Gbps, a common value in practice.

Each test run lasted 1,000 seconds and the TCP throughput was measured every 3 seconds. The first 3 seconds were discarded to eliminate setup variations. Results are shown in Table III.

TABLE III
AVERAGE TCP THROUGHPUT OBSERVED IN DIFFERENT CONDITIONS; RANGES CONSIDER A 99% CONFIDENCE INTERVAL.

Scenario	Bandwidth (Mbps)
No isolation, no attack	928.29 +/– 0.16
DCPortalsNg, no attack	928.21 +/– 0.22
No isolation, under attack	41.21 +/– 12.99
DCPortalsNg, under attack	910.11 +/– 0.28

 ISBN: 978-1-61208-318-6

Clearly, we can see the difference in the two cases. The system with no isolation suffers a loss of about 95% of the observed throughput. *DCPortalsNg* suffers just about 5% loss. With isolation, the UDP flow is blocked at the edge switches, not being delivered to other networks. The loss in this case is due to the overhead at the edge switch to drop the incoming UDP packets. Considering the system with no attacks, there is no statistical difference between *DCPortalsNg* and the standard switch scenario. That should be expected, considering the previous analysis of latency overheads.

IV. Related Work

As we mentioned in the introduction, this work is a continuation of our previous work in the area [5]. Compared to that work, *DCPortalsNg* provides a better integration with OpenStack, a simpler API to define the tenant networks, and an improved rewriting scheme.

Greenberg *et al.* [1] put together an interesting study about costs in a cloud computing datacenter. Among other observations, the network is identified as one of the major challenges in that context. Authors explicitly mention the dependence of current solutions on VLANs, and the problems associated with that practice.

One of the first initiatives towards a technique for virtual networks isolation was developed by a group at HP Labs [8], beginning with the definition of the concept of Trusted Virtual Domains (TVDs). Those would be logically isolated network sections, independent of the infrastructure topology. To implement that isolation, the authors implement a module internal to the virtual machines that is responsible for all processing related to network isolation. Two techniques for isolation, VLAN tagging and the EtherIP encapsulation, are compared. That work has a similar goal to *DCPortalsNg*, but the the solutions considered have scalability limitations and require intrusive modifications to the hypervisor. A longer comparative study by the same group mentions the MAC rewriting technique [2].

DCPortalsNg uses the SDN paradigm to solve the isolation problem. Pettit *et al.* [9] have already discussed the viability of such approach to datacenter networks, but did not present a concrete solution. Two applications of NOX (another SDN controller) to the datacenter were published previously, but they focused on implementing new network architectures and traffic control [3], [4]. Different from those solutions, *DCPortalsNg* does not require hardware with OpenFlow capabilities inside the network core, and focuses only on traffic isolation.

One work with very similar motivation to the one presented here is certainly Netlord, developed by Mudigonda *et. al.* [10]. In their paper, the authors use a solution based on encapsulation to achieve a similar traffic isolation without requiring special hardware in the network. However, the way their solution is implemented is quite different, using an extension of the Xen hypervisor specially developed for that goal. We believe that the use of Software Defined Networks is a more elegant and flexible approach and a determinant characteristic of our work. It simplifies implementation and offers a more flexible solution. *DCPortalsNg*, for example, works directly not only with Xen but with other hypervisors that use libvirt and Open vSwitch, such as KVM.

V. Conclusion and Future Work

This work presented *DCPortalsNg*, a system developed to provide traffic isolation for virtual networks in a virtualized datacenter environment. The system architecture and the implementation details were described, along with results that confirm the isolation provided. Evaluations also quantified the overheads during flow setup, which are noticeable but rare, and showed that during normal flow operational costs are negligible. Finally, we showed that the system can be effective at protecting tenants from denial-of-server attacks inside the datacenter network. As future work, we continue to improve the system. In particular, we are working on integrating *DCPortalsNg*, which provides network isolation, with Gatekeeper, a system designed to provide network traffic guarantees in a datacenter environment [11].

Acknowledgments

This work was partially sponsored by UOL, Fapemig, CNPq, and the National Institute of Science and Technology of the Web, InWeb (MCT/CNPq 573871/2008-6).

References

[1] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2009.

[2] S. Cabuk, C. I. Dalton, A. Eduards, and A. Fischer, "A comparative study on secure network virtualization," HP Laboratories, Tech. Rep. HPL-2008-57, 2008.

[3] 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, pp. 1–6.

[4] B. Heller *et al.*, "Ripcord: a modular platform for data center networking," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 457–458, 2010.

[5] R. V. Nunes, R. L. Pontes, and D. Guedes, "Virtualized network isolation using software defined networks," in *Proceedings of the 38th IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2013, pp. 700–703.

[6] 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*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–16.

[7] "Bitbucket amazon ddos attack," http://blog.bitbucket.org/2009/10/04/on-our-extended-downtime-amazon-and-whats-coming/ (retrieved: Dec 2013), 2012.

[8] 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*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 235–245.

[9] 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)*, ser. DC CAVES. Amsterdam: ITC, september 2010, pp. 1–7.

[10] 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*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 62–73.

[11] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proceedings of the 3rd conference on I/O virtualization*, ser. WIOV'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 6–6.

 ISBN: 978-1-61208-318-6