XenFlow: Seamless Migration Primitive and Quality of Service for Virtual Networks

Diogo Menezes Ferrazani Mattos, Otto Carlos Muniz Bandeira Duarte
Grupo de Teleinformática e Automação
Universidade Federal do Rio de Janeiro (COPPE/UFRJ)
Rio de Janeiro – RJ – Brasil
Email:{menezes,otto}@gta.ufrj.br

***Abstract*—Next generation network offers virtual networks on demand, each one with its own features and Quality of Service (QoS) requirements. Besides, live-migration provides a flexible and seamless topology remapping feature for virtual networks, but it is usually limited to a local area network. In this paper, we propose XenFlow, a hybrid virtualization system, based on Xen and OpenFlow technologies. XenFlow main goals are threefold. First, it provides a flexible virtual network migration primitive, as it deploys a Software Defined Networking between virtual machines, based on OpenFlow. Second, it provides a strong isolation of virtual networks, avoiding deny of service caused by interference of other virtual networks. Third, XenFlow offers inter-network and intra-network QoS provisioning by a consistent resource controller. We developed a prototype and our results show that the proposed system performs better than native mechanism of Xen virtual machine migration. XenFlow allows virtual router migration between different local area networks without creating tunnels or losing packets. Our experiments also show that resource usage controller meets QoS requirements and outperforms other techniques while it redistributes idle network resources.**

I. Introduction

The next generation network foresees a pluralist model, in which users open specific and customizable virtual networks on demand to comply their needs [1]. To this end, the pluralist network model requires a virtual network architecture capable of providing flexible topology mapping and secure and coherent Quality of Service (QoS) provisioning. The most promising techniques for the pluralist model are network virtualization, available through machine virtualization, and Software Defined Networking (SDN). Machine virtualization allows many different and isolated guests to share the same physical substrate. SDN allows programming network general purpose elements. Two important platforms for virtualizing and for providing Software Defined Networking are Xen [2] and OpenFlow [3]. Xen is a virtualization platform designed for server consolidation. Xen creates virtual environments that simulate entire physical machines, called virtual machines. OpenFlow is an Application Programming Interface (API), which implements network control in an independent and centralized node, called OpenFlow controller. The controller defines forwarding for all network nodes.

Both network virtualization and SDN decouple the network function from its physical realization [4]. Therefore, both of them introduce a new management primitive: the migration of virtual networks [5]. Migration is relocating the logical topology over the physical topology. Migration may be used in different contexts, such as, the maintenance of network nodes and provisioning energy efficient networks. Nevertheless, virtual link live-migration, that stands for remapping a virtual link on physical links, and reducing the time that a virtual node is unavailable during migration are still challenges of virtual network migration.

In this paper, we propose XenFlow, a hybrid network virtualization system that allows creation of highly programmable virtual networks. XenFlow supports seamless virtual network migration of both virtual network elements and links, without any packet losses or virtual topology changes. XenFlow also guarantees Quality of Service (QoS) provisioning and isolation between virtual networks through an efficient network isolation mechanism and a resource controller. XenFlow uses QoS parameters of virtual routers to manage available resources on physical substrate, and redistributes idle resources between virtual routers considering virtual router priorities. XenFlow applies the plane separation paradigm to enhance performance of virtual networks. Plane separation divides the routing function into two planes, the control plane and the data plane. The control plane is responsible for network control functions, such as routing calculation. The data plane is responsible for forwarding packets according to control plane policies and QoS parameters. The proposed system consists of hybrid virtualization architecture combing Xen and OpenFlow platforms to provide a programmable environment of virtual networking. Therefore, control planes of virtual networks are implemented in Xen virtual machines, while a shared data plane is deployed as an OpenFlow-compliant switch in each physical node.

The proposed system monitors resource usage of each virtual network and, based on these data, calculates the redistribution of idle resources between virtual networks. A XenFlow prototype was built to validate the system architecture. Our results show that the redistribution algorithm reduces idleness of network resources, when compared to other techniques, and assigns the reserved bandwidth for each virtual network in proportion to the amount hired by each network. Experiments also show that the system provides a robust migration primitive, as there are no packet losses or routing service interruptions during the virtual network migration. When we compare XenFlow migration and Xen virtual machine native migration, XenFlow shows zero packet losses, while Xen native migration loses a significant amount of packets and presented a longer period of control plane interruption.

The remainder of the paper is organized as follows. The discussion of the system design and its main components

are showed in Section II. Section III presents an analysis of experimental results. Section IV discusses related works. Conclusions and future works are discussed in Section V.

II. XenFlow Architecture

XenFlow provides a network virtualization system that enables isolating, providing Quality of Service, and migrating virtual networks. XenFlow ensures isolation of virtual networks since it provides address space isolation among virtual networks and it enforces resources sharing for each virtual network [6]. Thus, a virtual machine can only access virtual machines that belong to its virtual network and, also, a virtual network cannot use resources of other virtual networks. The system also offers Quality of Service through mapping parameters of Service Level Agreements, defined as control plane directives, to parameters of the data plane, controlling the resources usage of each virtual network. The basic resources controlled by XenFlow are processing, memory, and bandwidth of virtual networks, as those are the resources that can be locally controlled [7].

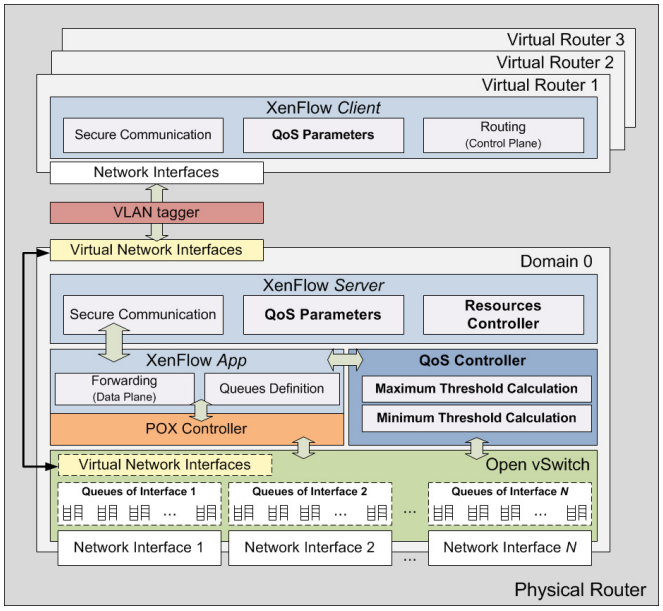


Figure 1. XenFlow node architecture. XenFlow Server runs on physical router that hosts a set of virtual routers running a XenFlow Client instance over each one. POX controller runs in a single centralized node. All other XenFlow nodes connect to it.

The system applies the plane separation paradigm to achieve higher performance on forwarding packets [2], [1]. Virtual machines act as the routers control plane, running routing protocols. Besides, data planes of all virtual routers run centrally in the Domain 0, which is a virtual machine with special privileges that directly accesses the physical network interfaces. This paradigm, however, forces all virtual networks to share the same data plane, violating the requirement of isolation between virtual environments.

XenFlow employs an OpenFlow switch as the routers' data plane. XenFlow manages the flows of each virtual router on a single OpenFlow switch on the privileged Domain 0. Therefore, resource isolation in XenFlow requires dedicated forwarding mechanisms to each virtual router in data plane and also control mechanisms. XenFlow isolates virtual networks associating each virtual router to a queue, or a set of queues, compliant with HTB (Hierarquical Token Bucket) control policies.

The XenFlow node architecture, shown in Figure 1, consists of a personal computer with commodity hardware running the Xen virtualization platform and a packet forwarding module compliant with OpenFlow Application Programming Interface (API). The packet forwarding module can be a software switch, like Open vSwitch [8]; a hardware-based switch; or a high-performance commercial OpenFlow solution. The virtual routers consist of Xen virtual machines running routing protocols. In this work, we use the extensible routing platform XORP (eXtensible Open Router Platform) [9] to deploy into virtual routers the main routing protocols and to allow the inclusion of new protocols. As the virtual routers may use legacy routing protocols, XenFlow can be incrementally deployable.Virtual routers run the *XenFlow Client* module, which checks for updates in its routing table and ARP table (Address Resolution Protocol). The routing table stores the information about routes calculated by the routing protocols, which run in the virtual router. ARP table stores the mapping of IP address to MAC addresses known by the virtual router. Collected information in the virtual machine is then sent to *XenFlow Server* module that runs in Domain 0. The *XenFlow Server* module generates information summaries for all virtual routers and sends them to *XenFlow App* application, creating a Routing Information Base for each virtual router. *XenFlow App* application runs on top of POX controller[1], which controls the Open vSwitch of Domain 0.

It is worth mentioning that whole process of communication between modules in XenFlow is encrypted and authenticated using the scheme of Public Key Infrastructure and follows the standard SSL 3.0 (Secure Socket Layer).

A. From Routing Information to Flow Mapping

XenFlow forwards data packets as follows. As a packet arrives at node data plane, XenFlow forwards it directly if the packet matches any flow in OpenFlow flow table. If the packet header does not match any flow, the packet header is sent to controller to define packet's next hop. When the controller receives a packet header, the *XenFlow App* identifies the Routing Information Base it has to consult regarding that packet header. The *XenFlow App* queries the Routing Information Base to infer the output interface and next hop, and then inserts a new flow in the flow table. Packets arrive at XenFlow nodes with the destination MAC address of the virtual router. Thus, this address has to be modified to the next hop MAC address and the source MAC address has to be set to the address of the virtual router output interface. The next hop MAC address is extracted of *ARP Table* copy on *XenFlow App*. This procedure also maps a virtual link into one or more physical links, since it may introduce layer-2 forwarding flows in flow table, as if two routers, in two different networks, had a common link between them.

[1]POX is an open source OpenFlow controller. It enables to develop Python applications to control an OpenFlow network. POX is available at http://www.noxrepo.org/pox/about-pox/.

B. Virtual Network Isolation

The main challenge of isolation between virtual networks is to use data plane primitives to tag which packets belong to each virtual network. Thus, XenFlow is able to route packets directly in the data plane. XenFlow access information about Ethernet, VLAN and IP layers, through the twelve fields of OpenFlow specification. The key idea of isolating virtual networks is to insert a VLAN tag in each packet that leaves a virtual machine and remove the tag as the packet goes to the virtual machine. Thus, a VLAN tagger acts between the virtual network interfaces and the OpenFlow switch. As a packet that enters or leaves a virtual machine, it is tagged or untagged with the VLAN ID of the virtual network that it belongs to. The VLAN tagger is configured at the VM creation time swith the identifier (ID) of the virtual network that each virtual network interface belongs to. The operation of the VLAN tagger is out of the virtual machines, thus it is transparent to virtual machines. The VLAN tagger is also responsible for ensuring that packets that do not belong to a given virtual network reaches other virtual networks, because the VLAN tagger drops packets that arrive to it, but do not have the VLAN tag with the correct identifier.

Nevertheless, routing packets occurs between different networks and, thus, between different VLANs. As *XenFlow App* identifies the output interface of the virtual router that the packet should be forwarded, it also identifies the output VLAN of a flow. Therefore, *XenFlow App* query which the marker is associated with the VLAN virtual interface network, in which the packet would be sent if it were actually sent by the virtual machine, retrieves the new VLAN identifier of the packet and adds the new flow in plan OpenFlow data. The new flow is introduced according to the fields of the packet that triggered the calculation and the actions associated with this new flow are changing the source MAC addresses and destination of the package, changing the VLAN identifier of the package and, finally, forward the packet to the output port proper.

C. Virtual Router Migration

In XenFlow, a virtual link can be mapped into one or more physical links. The routing function is performed by a flow table dynamically controlled by POX and the topology of the virtual network is decoupled from its physical realization [4]. As a consequence, migrating virtual routers, shown in Figure 2, consists of three steps: migration of control plane, reconstruction of data plane, and migration of virtual links. The control plane is migrated between two physical network nodes through the live-migration mechanism of conventional Xen virtual machines [10]. Then, the reconstruction of data plane is performed as follows. *XenFlow Client* connects to the Domain 0 daemon, *XenFlow Server*, and sends all routes that are known by the virtual machine. When the *XenFlow Client* detects a connection disruption with the Domain 0 caused by a migration, the application reconnects, now on the new server in which the virtual router is hosted, and sends all information about its routing and ARP tables. Upon receiving such information, Domain 0 reconfigures the *XenFlow App*, running over POX controller, to execute the data plane according to the control plane of the migrated virtual router. Thus, all packets that arrive at the physical machine are handled according to the control information computed by the control plane that was previously migrated. Note that these packets are addressed to the migrated virtual router and that the physical machine is the one which the virtual network router was migrated. After migration of the control plane and reconstruction of the data plane, links are migrated. The links migration occurs in OpenFlow switches (Open vSwitch) instantiated in Domain 0 of physical servers and other switches in the network. Link migration occurs in order to create a switched path between all neighbors of the migrated virtual router, in the virtual topology, to the physical router that hosts the virtual router after migration. Thus, the migrated virtual router sends an `ARP reply` packet with a predefined destination MAC address (`AA:AA:AA:AA:AA:AA`). In a XenFlow network, this MAC address is reserved and this packet has priority in being processed over other packets sent on the network. Predefined `ARP reply` packets are processed at the first switch that they arrive and, then, they are dropped. As a consequence, the migrated virtual router announces where it is available from now on using this special packet. This procedure updates the location of a virtual router in the network associating link migration with a dual behavior, router and switch, of XenFlow nodes. The dual behavior results in a migration primitive of virtual routers that has no packet loss or interruption of packet-forwarding services.

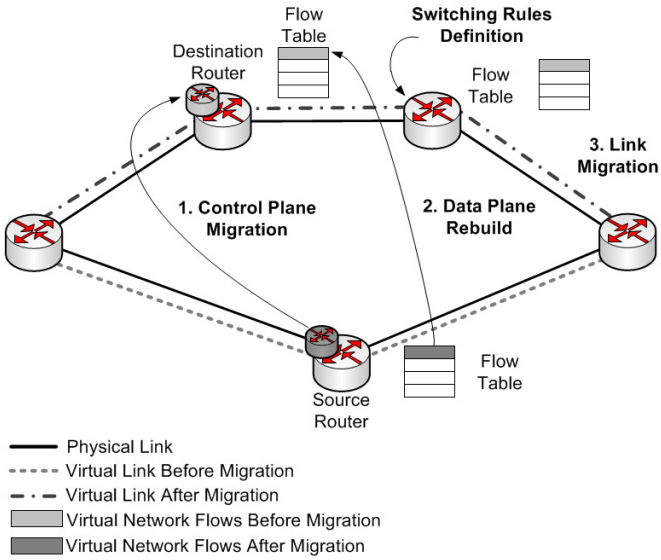


Figure 2. Three steps of the XenFlow virtual topology migration. XenFlow first migrates the virtual machine running routing protocols. Secondly, it rebuilds the data plane based into control plane information. Afterwards, XenFlow migrates links sending a predefined `ARP Reply` message.

D. Resource Isolation

XenFlow resource control focus on memory, processor, and bandwidth per network interface used by each virtual router. Memory isolation is done in two levels. Firstly, Xen memory isolation mechanism limits the physical memory dedicated for each virtual router. Besides, XenFlow modules limits the number of routes reported to *XenFlow Server* and the number of flows inserted into flow tables by *XenFlow App*. As XenFlow may reject some routes or flows, a default route and a wildcard flow are inserted into XenFlow modules to avoid packet losses. Processor usage isolation is deployed as a static control. Xen

native scheduling mechanism is enough to ensure processor usage isolation between virtual routers [1]. Nevertheless, plane separation introduces a processing overhead on Domain 0, which is not controlled by Xen scheduling mechanism, but, as it depends on packet forwarding rate, it may be controlled by bandwidth limitation mechanisms. Bandwidth control for each virtual router is done by associating virtual machines to queues in OpenFlow forwarding mechanism. OpenFlow supports creating isolated queues in each network interface. Each queue has two tresholds: the minimum and the maximum bandwidth values. The minimum value is the minimum bandwidth guaranteed for that queue on link sharing. The maximum value is the ceiling transmission rate that a queue can take.

Queue control applied by the OpenFlow, however, is static and does not consider the priority of queues to redistribute the idle link capacity [11]. Therefore, one of the contributions of this paper is a queue controller that performs the redistribution of idle capacity of a link in accordance with the priority of each queue and inversely proportional to how much bandwidth a virtual network is using beyond its hired capacity.

E. Bandwidth Control

The queue controller acts on two levels. The first level is when the total capacity of the link is greater than the aggregated minimum guaranteed values of all queues within a link. In this case, the controller acts on the minimum value guaranteed for each queue. The second case is when each queue has its minimum capacity defined, but the total usage of the link is smaller than the sum of the minimum guaranteed bandwidth values of all queues. In this case, the controller acts on the maximum bandwidth allowed for each queue. In both cases, the redistribution considers the priority of each network, and, in XenFlow, the priority of a network is the relative value of minimum bandwidth guaranteed for a given network divided by the sum of the amounts of bandwidth guaranteed for all networks sharing the same link. The controller measures the usage of each queue regularly every t_{med} seconds, stores the last k_{med} values for each queue, and calculates the redistribution of idle capacity.

The minimum threshold control redistributes resources not allocated to queues according to two criteria: (i) a virtual router that hires a greater routing capacity gets **greater weight** in the redistribution of idle resources; (ii) the virtual router that is using relatively more resources than it has contracted gets a **lower weight** in the redistribution of idle resources.

The control of the maximum bandwidth allowed for each virtual network aims to optimize the use of the link idle capacity according to the priority of each virtual network. We calculate the link idle capacity adding the average load of all queues in a link and, after that, subtracting it from the value of the link bandwidth capacity. The resulting value is the amount of resources that should be redistributed. Redistribution, given by

$$max_{queue} = min_{queue} + C * \frac{min_{queue}}{\sum_{i}^{i \in Link} min_i}, \qquad (1)$$

where C is the link idle capacity, ensures that resources allocated for a queue, and not used, are allocated proportionately to the other queues. The redistribution is proportional to the minimum bandwidth hired by each queue.

F. Quality of Service Provisioning

Bandwidth controller provides Quality of Service in XenFlow. The bandwidth controller guarantees a minimum QoS level for each virtual network and also guarantees that the idle resources are redistributed accordingly to the priority of each virtual network. Moreover, depending on how mapping between virtual networks and queues is done, it is possible to define two different levels of QoS in virtual networks: (i) **Inter-network Quality of Service** is reached when each virtual network is associated with a single queue. In this case, the QoS parameters are only defined for the virtual network, allowing differentiated services of a virtual network from other services; (ii) **Intra-network Quality of Service** is reached when a set of queues is associated with a single virtual router. Each queue has its own QoS parameters and, depending on the configuration of each virtual router, a set of flows is mapped to each queue of the virtual router. Mapping a set of flows in a queue allows guarantying an amount of bandwidth for sensitive flows within a single virtual network. Intra-network QoS is a generalization of inter-network QoS for a set of flow and, therefore, they are not mutually exclusive.

Mapping flows to queues is accomplished by *XenFlow App* that runs on top of POX. Definiting which queue is dedicated to each virtual router, and, when considering intra-network, what flow characteristics are considered to map each flow in a queue, are configured in *XenFlow Server* module that communicates with *XenFlow App*.

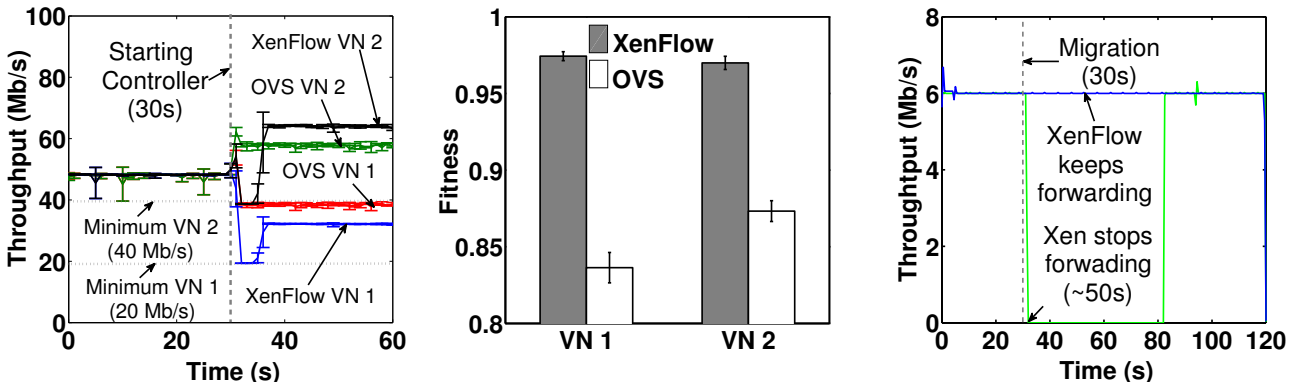
III. Experimental Results

We prototype XenFlow using Xen 4.0 to provide control plane and the Open vSwitch 1.2.2 to provide data plane forwarding functions. The Open vSwitch [8] is configured to act as OpenFlow switch, controlled by POX. The application that performs the plane separation and redirects packets to appropriate queues in the data plane is written in Python and runs on top of POX. The Quality of Service controller, which performs the setting of maximum and minimum thresholds of each queue, was written in Python and interacts directly with the configuration interfaces of Open vSwitch. The tools `Iperf`[2] and `Tcpdump`[3] were employed to measure system performance.

Four personal computers compose the experimental scenario of the experiments. All computers are equipped with Intel Core 2 Quad 2.4 GHz and 4 GB of RAM. Each computer has at least three network interfaces all of which are configured to operate at 100 Mb/s, since there was 1 Gb/s and 100 Mb/s interfaces. XenFlow server hosts three virtual machines that perform as routesr. Each virtual machine is configured with one virtual CPU, 128 MB of RAM and runs Debian Linux 2.6-32-5. The virtual machines run routing protocols over XORP [9], however, while running tests, we configured static routes. The results presented in this section are averages of 10 rounds of each experiment, with a confidence interval of 95%.

[2]http://iperf.sourceforge.com.

[3]http://www.tcpdump.org.



(a) Controller starts after 30 s. (b) Network fitness to SLA. (c) Virtual router migration occurs at 30 s.

Figure 3. XenFlow proportionally redistribute unused resources with two virtual networks (VN 1 and VN 2). a) Comparison between XenFlow and Open vSwitch resource sharing. b) QoS controller fitness to the proportional redistribution statement. c) XenFlow migrates virtual router without stopping forwarding.

The redistribution of idle resources of a link must respect the priority of each virtual network. To evaluate how fair XenFlow respects the priority of each virtual network, we defined the metric

$$fitness = 1 - \left| 1 - \frac{Bandwidth_{obtained}}{Bandwidth_{expected}} \right|, \qquad (2)$$

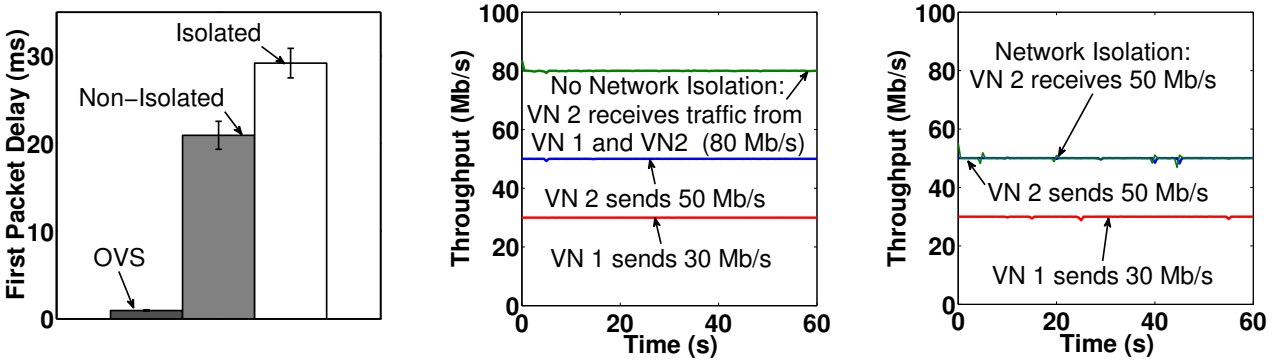
which calculates the percentage of accuracy of the resource redistribution according to the priority of each network. The `fitness` metric determines how close the bandwidth, destined to a virtual network, is to its optimal calculated value.

The first experiment verifies the efficiency of the QoS controller, considering the proportional and differential redistribution of resources, referenced in Figure as `XenFlow`, and compares the results with the distribution of the idle resources on Open vSwitch, referenced as `OVS`. We set up Open vSwitch to direct the flows of each virtual network to a particular queue and set up a guaranteed minimum bandwidth of each queue, according to the definition of the network, and the maximum ceiling value, to be a fairly comparison, was configured for the link maximum bandwidth, 100 Mb/s. In XenFlow, we defined the minimum bandwidth reserved for each queue and the maximum bandwidth is set by the controller. Figure 3(a) shows the experiment for two UDP flows of 1472 B packets, for two different virtual networks, `VN 1` and `VN 2`, with a minimum of 20 Mb/s and 40 Mb/s, respectively. The dynamics of the experiment is as follows. The experiment starts at 0 s. At the beginning of the experiment, no bandwidth limit is set and the link is fully shared by the two networks. In 30 s, it is set the minimum and maximum thresholds of the queues, on Open vSwitch setting up, and the controller of QoS is started on XenFlow setting up. The experiment ends at 60 s.

Figure 3 shows the effectiveness of the proportional and differentiated redistribution of resources to Service Layer Agreement specified in XenFlow approach. The higher minimum bandwidth network is forwarded by higher priority queue and, therefore, accesses a larger slice of bandwidth allocation of the idle resource portion of the link. Native Open vSwitch approach shares free resources equally divided among all queues. Figure 3(b) shows the `fitness`, after defining the thresholds of each queue. The XenFlow `fitness` is around 14% higher than the Open vSwitch `fitness`, reaching 97%. Therefore, the XenFlow QoS controller is more able to provide Quality of Service to priority flows than simply applying OpenFlow QoS primitives, implemented by the Open vSwitch.

Our next experiment evaluates XenFlow live migration. Two physical machines forward packets, while the two other machines generate and receive packets. A virtual machine performs as a router. The experiments use two additional machines to generate or receive packets, each one communicates with both physical routers. The experiments consist of virtual router forwarding a 6 Mb/s UDP flow, with 1472 B packets, which is the most common size of Ethernet MTU (Maximum Transmission Unit). While virtual router is forwarding packets, after 30 s, it is migrated from source physical machine to the destination one, using both Xen native migration and our proposed XenFlow migration. Results, shown by presents our results of the experiment, indicate that the migration performed by native-Xen application, referenced in figure as `Xen`, implies the interruption of data forwarding for approximately 50 s. This interruption occurs due to deactivation of the virtual machine while its memory is transferred from source physical server to destination physical server. Thus, as native Xen necessarily forwards packets throw the virtual machine, stopping the virtual machine results in the interruption of packet forwarding. Migration is only possible without packet loss when applying plane separation paradigm, because packets are forwarded by Domain 0, data plane. Therefore, while the virtual machine is migrated, data plane remains active in source physical server, forwarding packets thorn previously established connections. XenFlow employs plane separation paradigm and Figure 3(c) shows that migration of a virtual router occurs with no packet loss, referenced in figure as `XenFlow`.

The evaluation of the delay introduced by XenFlow forwarding mechanism considers the round-trip time (RTT) of ICMP Echo Request and Echo Reply. Figure 4(a) compares the delay introduced by XenFlow, referred to as `Isolated`; routing without adding VLAN tag, referred to as texttt Non-Isolated; and the delay of the Open vSwitch acting as software switch, referred to as `OVS`. Open vSwitch configuration refers to packet switching between physical node interfaces. Although XenFlow engine uses the Open vSwitch, it also realizes the plane separation based on the `XenFlow App`,



(a) Delay of forwarding the first packet of a flow. (b) Non-isolated virtual network. (c) Isolated virtual network.

Figure 4. Packet forwarding for two virtual networks, VN 1 and VN 2. Both have the same IP Address space. a) Comparison of delay introduced by tagging packets; b) non-isolated networks - both networks receive packets sent to the common IP address. VN 2 receives the sum of the rates (80 Mb/s) sent both VN 1 (30 Mb/s) and VN 2 (50 Mb/s); c) isolated networks - each virtual network receives only packets that are intended to its. VN 2 receives a rate of 50 Mb/s.

controling the OpenFlow data plane. The experiment only considers the packet delay introduced by the instantiation of the flow in the OpenFlow data plane. Figure 4(a) indicates that the delay introduced `XenFlow App`, even without isolation, is of the order of 20 ms. When handling packets and defining the VLAN in which the packet should be forwarded, the delay of the first packet arises to approximately 30 ms. The delay, however, is related to the forwarding the first packet of each flow for POX, which is the usual flow installation process in OpenFlow. Routing the following packets introduces the same delay for all three approaches, which shows that the forwarding delay is due only to the forwarding mechanism.

The last experiment verifies the effectiveness of the isolation of virtual networks, when an attacker tries to eavesdrop on communications from another virtual network or inject traffic on another network. The key idea of this experiment is to define two virtual networks, and then we try to inject traffic from one network into the other. The Virtual Network 1 (VN 1) consists of a virtual machine hosted on Physical Router 1. The Virtual Network 2 (VN 2) consists of a virtual machine hosted on Physical Router 1 and another hosted on Router Physical 2. Both nodes on Router Physical 1 are sending 1500 B UDP packets. All virtual machines were configured to belong to the same IP address space and send data to the same destination IP. However, as the virtual networks are isolated, it is expected that the flow of the VN 1 does not interfere with the flow of the VN 2. Figure 4(b) shows that the scenario without our isolation mechanism, the receiving node of VN 2 receives packets from both, VN 2 and VN 1, indicating the absence of isolation in IP address spaces, and being able to eavesdrop flows from other virtual networks. On the other hand, Figure 4(c) demonstrates that XenFlow isolates the address space of each virtual network because the traffic of VN 1 does not interfere with traffic from VN 2, since the VN2 receiver only receives packets belonging to its own network.

IV. Related Work

Virtualization technology enables the Future Internet pluralist approach of several virtual networks running over the same physical substrate. Besides, there are proposals for SDN that stands for programming general purpose network devices. Isolation, QoS provisioning, and virtual topologies migration, however, are still being challenges [1].

Distributed Overlay Virtual Ethernet (DOVE) [12] is a proposal of network virtualization that provides isolation by using a network identifier that is added to the envelop DOVE header, creating an overlay network. Isolation is also achieved using VXLAN encapsulation [13]. VXLAN also adds to each Ethernet frame an outer Ethernet header, followed by an external IP, UDP and VXLAN headers. *Network Virtualization Generic Routing Encapsulation* (NVGRE) [14] also encapsulates to allow multitenancy in public or private clouds. Both VXLAN and NVGRE use 24 bits to identify the virtual network that a frame belongs to. Nevertheless, these proposals create an overlay network that interconnects the nodes of the virtual network. Such proposals fail to apply the paradigm of planes separation, as these proposals do not provide a mechanism to directly forward packets between different networks in the data plane. On the other hand, XenFlow isolates virtual networks in link layer, even in the plane separation scenario. The proposal inserts a VLAN tag in packets of each virtual network, eliminating the need to create an overlay network. An major contribution of the paper is to perform the isolation of virtual networks through primitive of the data plane, like VLAN tagging. It is worth mentioning that XenFlow keeps all the packets at the link layer, Ethernet, since there is no encapsulation of frames per network layer protocols, leaving all fields of the original package of the virtual network visible to the OpenFlow controllers, while other proposals hide above-layer fields of the original packet, as it is encapsulated in a new packet whose destination are the edges of the created tunnels.

Houidi *et al.* propose an adaptive system for provisioning virtual networks [15]. The key idea is to provide resources on demand for virtual networks, as soon it detects service degradation of each virtual network, or after a resource failure. OMNI (OpenFlow Management Infrastructure) [16] also provides Quality of Service to OpenFlow networks [3]. Actions taken by OMNI, however, are restricted to flow migrations. Besides, Kim *et al.* propose mapping QoS parameters on resources available on OpenFlow switches, such as queues and rate limiters [11]. The proposal main goal is to provide QoS to scenarios in which the physical infrastructure is shared by virtual networks with different workloads. Nevertheless, the

control of QoS parameters and QoS mapping are centralized on the OpenFlow controller node.

Wang *et al.* present OpenFlow as a network management tool [17]. Wang *et al.* propose a load balancer based on programming low cost OpenFlow switches to multiplex requests among different server replicas. The proposal, however, does not guarantee the reservation of resources, nor QoS of flows. Hao *et al.*, in their turn, present the infrastructure VICTOR (*Virtually Clustered Open Router*) [18] which is based on creating a cluster of datacenters via a virtualized network infrastructure. The central idea of this approach is to use the OpenFlow as the basic network infrastructure of datacenters to allow moving a virtual machine from one physical location to another.This proposal optimizes the datacenter network usage performing server migrations, but it does not guarantee Quality of Service of each flow, and also does not isolate the use of resources from different virtualized servers. XenFlow, in turn, ensures isolation of resources shared by the virtual routers. It performs efficient sharing of idle resources and supports applying to Service Level Agreements (SLAs), such as guarantying a minimum bandwidth to a virtual router.

V. Conclusion

In this paper, we propose the XenFlow system, which is a hybrid virtualization and Software Defined Networking system. XenFlow implements a commodity hardware routing architecture, combining Xen virtualization technology and OpenFlow API. Our routing architecture is backward compatible and incrementally deployable, as the current protocols can communicate with XenFlow virtual routers running current protocols. Likewise, XenFlow employs OpenFlow switches to forward traffic, which enables the gradual adoption of SDN techniques. XenFlow isolates the resource consumption of each virtual network and also provides Quality of Service (QoS). XenFlow also implements zero packet loss virtual router migration and eliminates the need of tunnels or external mechanisms for link migrations. We developed a prototype and analyzed it. Our results show that the XenFlow has no data plane downtime, while native Xen migration introduces up to 30 s of data plane downtime, which confirms the zero packet loss virtual router migration of XenFlow. Evaluating the QoS capability, the XenFlow resource controller reached the maximum utilization of the link and achieved an efficient proportional distribution of resources of approximately 97%, which is 14% higher than the result obtained with the Open vSwitch. At last, our experiments also confirm that XenFlow is able to isolate two virtual networks even when both networks share the same IP address space. As a future work, the VLAN tag will be replaced by a Multiprotocol Label Switching (MPLS) tag, whose semantics of virtual circuit identifier will be overloaded and it will address up to 8 million possible virtual networks, using 23 available bits in MPLS header. We will also develop optimization algorithms for allocating virtual networks on the physical network infrastructure.

Acknowledgments

The authors would like to thank FINEP, FUNTTEL, CNPq, CAPES, FAPERJ and UOL for their financial support.

References

[1] I. M. Moraes, D. M. Mattos, L. H. G. Ferraz, M. E. M. Campista, M. G. Rubinstein, L. H. M. Costa, M. D. de Amorim, P. B. Velloso, O. C. M. Duarte, and G. Pujolle, "FITS: A flexible virtual network testbed architecture," *Computer Networks*, no. 0, pp. –, 2014.

[2] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Towards high performance virtual routers on commodity hardware," in *Proceedings of the 2008 ACM CoNEXT*. ACM, 2008, pp. 1–12.

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

[4] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," in *ACM SIGCOMM PRESTO'09*. ACM, 2010, p. 8.

[5] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 231–242, 2008.

[6] M. Bari, R. Boutaba, R. Esteves, L. Granville, M. Podlesny, M. Rabbani, Q. Zhang, and M. Zhani, "Data center network virtualization: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 2, pp. 909–928, 2013.

[7] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," Tech. Rep. OPENFLOW-TR-2009-01, OpenFlow Consortium, Tech. Rep., 2009.

[8] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," *Proc. HotNets*, Oct. 2009.

[9] M. Handley, O. Hodson, and E. Kohler, "XORP: An open platform for network research," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 53–57, 2003.

[10] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[11] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S. Lee, and P. Yalagandula, "Automated and scalable QoS control for network convergence," *Proc. INM/WREN*, Apr. 2010.

[12] K. Barabash, R. Cohen, D. Hadas, V. Jain, R. Recio, and B. Rochwerger, "A case for overlays in dcn virtualization," in *Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching*. ITCP, 2011, pp. 30–37.

[13] Y. Nakagawa, K. Hyoudou, and T. Shimizu, "A management method of ip multicast in overlay networks using openflow," in *Proceedings of the first workshop on Hot topics in software defined networks*, ser. HotSDN '12. Helsinki, Finland: ACM, 2012, pp. 91–96. [Online]. Available: http://doi.acm.org/10.1145/2342441.2342460

[14] M. Sridharan, K. Duda, I. Ganga, A. Greenberg, G. Lin, M. Pearson, and P. Thaler, "NVGRE: Network Virtualization using Generic Routing Encapsulation," NVGRE, Internet Engineering Task Force, Feb. 2013. [Online]. Available: http://tools.ietf.org/html/draft-sridharan-virtualization-nvgre-02

[15] I. Houidi, W. Louati, D. Zeghlache, P. Papadimitriou, and L. Mathy, "Adaptive virtual network provisioning," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*. ACM, 2010, pp. 41–48.

[16] D. Mattos, N. C. Fernandes, V. Costa, L. Cardoso, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "OMNI: OpenFlow MaNagement infrastructure," in *2011 International Conference on the Network of the Future (NoF'11)*, Paris, France, Nov. 2011, pp. 52–56.

[17] R. Wang, D. Butnariu, and J. Rexford, "Openflow-based server load balancing gone wild," in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*. USENIX Association, 2011, pp. 12–12.

[18] F. Hao, T. Lakshman, S. Mukherjee, and H. Song, "Enhancing dynamic cloud-based services using network virtualization," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. ACM, 2009, pp. 37–44.