Annals of Telecommunications manuscript No.
(will be inserted by the editor)

Virtual Networks: Isolation, Performance, and Trends

Natalia C. Fernandes · Marcelo D. D. Moreira · Igor M. Moraes · Lyno Henrique G. Ferraz · Rodrigo S. Couto · Hugo E. T. Carvalho · Miguel Elias M. Campista · Luís Henrique M. K. Costa · Otto Carlos M. B. Duarte

Received: date / Accepted: date

Abstract Currently, there is a strong effort of the research community in rethinking the Internet architecture to cope with its current limitations and support new requirements. Many researchers conclude that there is no one-size-fits-all solution for all of the user and network-provider needs and, thus, advocate for a pluralist network architecture, which allows the coexistence of different protocol stacks running at the same time over the same physical substrate. In this paper, we investigate the advantages and limitations of the virtualization technologies for creating a pluralist environment for the Future Internet. We analyze two types of virtualization techniques, which provide multiple operating systems running on the same hardware, represented by Xen, or multiple network flows on the same switch, represented by OpenFlow. First, we define the functionalities needed by a Future Internet virtual network architecture, and how Xen and OpenFlow provide them. We then analyze Xen and OpenFlow in terms of network programmability, processing, forwarding, control, and scalability. Finally, we carry out experiments with Xen and OpenFlow network prototypes, identifying the overhead incurred by each virtualization tool by comparing it with native Linux. Our experiments show that OpenFlow switch forwards packets as well as native Linux, achieving similar high forwarding rates. On the other hand, we observe that the high complexity involving Xen virtual machine packet forwarding limits the achievable packet rates. There is a clear tradeoff between flexibility and performance, but we conclude that both Xen and OpenFlow are suitable platforms for network virtualization.

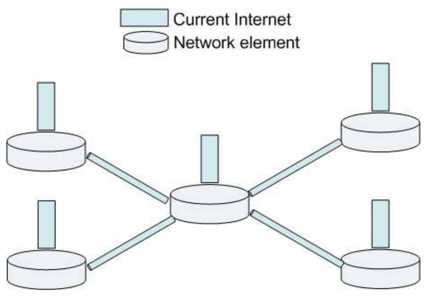
Keywords Future Internet · Virtual networks · Routing

This work was supported by CNPq, CAPES, FAPERJ, FUJB, FINEP, and FUNTTEL.

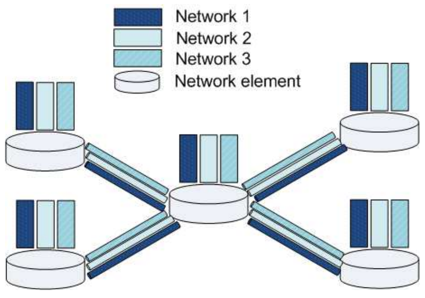
N. C. Fernandes · M. D. D. Moreira · I. M. Moraes · L. H. G. Ferraz · R. S. Couto · H. E. T. Carvalho · M. E. M. Campista · L. H. M. K. Costa · O. C. M. B. Duarte
Grupo de Teleinformática e Automação (GTA)
Universidade Federal do Rio de Janeiro (UFRJ)
Rio de Janeiro, Brazil
Tel.: +55-21-2562-8635
E-mail: {natalia,marcelo,igor,lyno,souza,hugo,miguel,luish,otto}@gta.ufrj.br

1 Introduction

The Internet is a great success with more than one billion users spread over the world. The Internet model is based on two main pillars, the end-to-end data transfer service and the TCP/IP stack [2]. Indeed, those two pillars guarantee that the network core is simple and transparent, while all the intelligence is placed on the end systems. This architectural choice makes it easy to support new applications, because there is no need to change the network core. On the other hand, this model ossifies the Internet, making it difficult to solve structural problems like scalability, management, mobility, and security [6]. Today, there is a rough consensus that upgrade patches are not enough to meet current and future requirements. Then, the Internet must be reformulated to provide a flexible infrastructure that supports innovations in the network, which is being called the Future Internet [3, 6]. We divide the models for the Future Internet into two types: monist, described in Fig. 1(a), and pluralist, described in Fig. 1(b) [1]. According to the monist model, the network has a monolithic architecture that is flexible enough to provide support to new applications. On the other hand, the pluralist approach is based on the idea that the Internet must support multiple protocol stacks simultaneously. Hence, the pluralist model establishes different networks, according to the needs of network applications. A characteristic in favor of the pluralist model is that it intrinsically provides compatibility with the current Internet, which can be one of the supported protocol stacks. Other specialized networks could be used to provide specific services, such as security, mobility, or quality of service. This is a simpler approach than trying to design a network that could solve all the problems that we already know in the network, as well as all the other problems that we still do not know, as suggested by the monist model.



(a) Current monist architecture model. Only one protocol stack running over the physical substrate.



(b) Pluralist architecture model. Different protocol stacks running, at the same time, over the same physical substrate.

Fig. 1 Models for the monist and pluralist network architectures.

All pluralist proposals are based on the same idea that multiple virtual networks should run over the same physical substrate [9], even though they differ in packet formats, addressing schemes, and protocols. Indeed, the virtual networks share the same physical medium, but run different protocol stacks independently from each other.

This paper addresses the issue of sharing the network physical substrate among different virtual networks. We analyze two of the main approaches for virtualizing the physical network, Xen [7] and OpenFlow [12], and discuss the use of these technologies for running virtual networks in parallel. The main objective of this paper is to investigate the advantages and limitations of the network virtualization technologies for creating a virtual environment that could be used as the basis of a pluralist architecture for the Future Internet.

The performance of Xen and OpenFlow in personal computers used as routers/ switches has been previously evaluated. Egi *et al.* [7] evaluate the performance of different forwarding schemes using Xen. Authors observe that forwarding packets using virtual machines severely impact system performance. Therefore, they evaluate the forwarding performance in a specialized domain, improving the performance of Xen to near non-virtualized environments. Egi *et al.* argue that each virtual machine requires CPU time to process its operations. Therefore, the number of virtual-machine context switches increases proportionally to the number of instantiated virtual machines, decreasing the performance of the whole system. Mateo analyzes packet forwarding using OpenFlow [11]. He investigates the network aggregated throughput under normal and saturated conditions. In addition, he also analyzes different OpenFlow configurations. Mateo shows that OpenFlow throughput can be as high as non-virtualized environments. Nevertheless, this performance is affected by the flow table size. He shows that flow tables configured using a hash algorithm reaches high aggregated throughput even under heavy traffic conditions.

The contributions of this paper are threefold. First, we establish the functionalities and primitives required by a virtual network architecture and show how Xen and OpenFlow satisfy such requirements. We then provide a detailed comparison of Xen and OpenFlow virtualization models, showing how they impact network scalability, programmability, processing, forwarding, control, and management. We show that a network virtualization model using Xen is a powerful and flexible solution, because the whole network element can be virtualized into virtual slices that have total control of hardware components. This powerful flexibility, however, impacts the packet forwarding performance [7]. A solution to this problem seems to be the sharing of data planes among virtual slices. OpenFlow follows this approach by defining a centralized element that controls and programs the shared data plane in each forwarding element. Finally, we carry out experiments to compare Xen and OpenFlow performance acting as a virtualized network element (router/switch) in a personal computer. Both Xen and OpenFlow are deployed in a Linux system. Hence, we use native Linux performance as a reference to measure the overhead introduced by each virtualization tool. We analyze the scalability of Xen and OpenFlow with respect to the number of parallel networks. Our experiments show that, using shared data planes, Xen and OpenFlow can multiplex several virtual networks without any measurable performance loss, comparing with a scenario where the same packet rate is handled by a single virtual network element. Delay and jitter tests show similar results, with no measurable overhead introduced by network virtualization tools, except for the case in which the traffic is forwarded by Xen virtual machines. Even in this worst case, which presents per-hop delays of up to 1.7 ms, there is no significant impact on real-time applications such as voice over IP (VoIP). Finally, our forwarding performance experiments show that OpenFlow switch forwards packets as well as native Linux, achieving about 1.2 Mp/s of packet forwarding rate without any packet loss. On the other hand, we observe a high complexity involving Xen virtual machine packet forwarding that limits the forwarding capacity to less than 200 kp/s. Differently from previous work, we also analyze how the allocation of processing resources affects Xen forwarding perfomance.

Based on our findings, we conclude that both Xen and OpenFlow are suitable platforms for network virtualization. Nevertheless, the tradeoff between flexibility and performance must be considered in the Future Internet design. This tradeoff indicates that the use of shared data planes could be an important architectural choice when developing a virtual network architecture.

The remainder of this paper is structured as follows. In Section 2, we discuss the approaches for network virtualization according to data plane and control plane structure. In Section 3, we describe both Xen and OpenFlow, and in Section 4, we discuss the use of these technologies for network virtualization. We describe the performance test environment and the obtained results in Section 5. Finally, in Section 6, we present our conclusions.

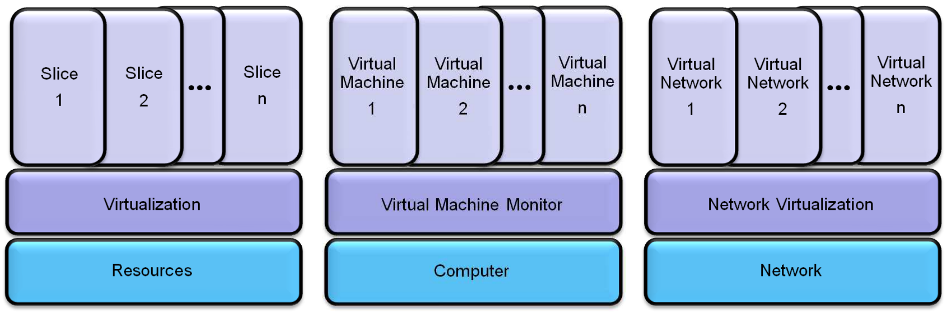
2 Approaches for Network Virtualization

We consider virtualization as a resource abstraction that allows slicing a resource into several slices, as shown in Fig. 2. This abstraction is often implemented as a software layer that provides "virtual sliced interfaces" quite similar to the real resource interface. The coexistence of several virtual slices over the same resource is possible because the virtualization layer breaks the coupling between the real resource and the above layer. Fig 2 shows two examples of virtualization: computer virtualization and network virtualization. The computer virtualization abstraction is implemented by the so-called Virtual Machine Monitor (VMM), which provides to virtual machines (VMs) an interface (i.e., the hardware abstraction layer) quite similar to a computer hardware interface, which includes processor, memory, input/output devices, etc. Thus, each virtual machine (VM) has the impression of running directly over the physical hardware, but actually the physical hardware is shared among several VMs. We call slicing this kind of resource sharing, because the virtual machines are isolated: one VM cannot interfere with other VMs. Computer virtualization is widely used in datacenters to allow running several servers in a single physical machine. This technique saves energy and reduces maintenance costs, but flexibility is the most important virtualization feature, because each virtual machine can have its own operating system, application programs, configuration rules, and administration procedures. The flexibility of running whatever is desired into virtual slices, such as different and customized protocol stacks, is the main motivation of applying the virtualization idea to networks [1]. As shown in Fig 2, network virtualization is analogous to computer virtualization, but now the shared resource is the network.

2.1 Network Virtualization for Accomplishing the Pluralist Approach

Network virtualization allows to *instantiate/delete* and *monitor* virtual networks and also to *migrate* network elements and *set* its resource-allocation parameters. Such functionalities make the network virtualization a suitable technology for creating multiple virtual networks and, as a consequence, for supporting the pluralist approach, because several requirements are satisfied, as explained below.

Functionality 1: Creation of Multiple Customized Networks - In a pluralist architecture, we have multiple networks running in parallel. The *instantiate* primitive can be used to instantiate virtual network elements, such as virtual routers and/or



(a) The concept of slicing resources by using virtualization. (b) Virtual slices on a computer harware. (c) Virtual slices on a network.

Fig. 2 Obtaining "sliced" resources by means of virtualization for different scenarios.

virtual links, and, therefore, multiple virtual networks can be rapidly deployed and run simultaneously. Each virtual network has its own protocol stack, network topology, administration policy, etc. This enables network innovation and new business models [9]. With network virtualization, a service provider can allocate an end-to-end virtual path and instantiate a virtual network tailored to the offered network service, e.g. a network with quality-of-service (QoS) guarantees. Hence, new services can be easily deployed and new players can break the barrier to enter in the network service market.

Functionality 2: Flexible Management - The network virtualization layer breaks the coupling between the logic used to construct the forwarding table and the physical hardware that implements the packet forwarding task [17]. Therefore, the *migrate* primitive allows moving a virtual network element from a physical hardware to another, without changing the logical/virtual network topology. In addition, traffic engineering and optimization techniques can use the *migrate* primitive to move virtual network elements/links along the physical infrastructure in order to minimize energy costs, distance from servers to specific network users, or other objective functions.

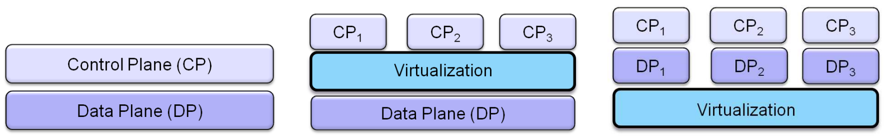
Functionality 3: Real-time Control - The virtual networks architecture also supports real-time control of virtual network resources because resource-allocation parameters can be *set* for each virtual network element (router, switch, link, gateway, etc.). We can set the allocated memory, bandwidth, maximum tolerated delay, etc. Even specific hardware parameters can be set, for instance, the number of virtual processors, priority of processor usage in a contention scenario, etc. Thus, we can dynamically adapt the resources allocated to each virtual network according to the current network condition, number of users, priority of each virtual network, service level agreements (SLAs), etc.

Functionality 4: Monitoring - Network virtualization also comes with a set of monitoring tools required to measure variables of interest, such as available bandwidth, processor and memory usage, link and end-to-end delay, etc. The *monitor* primitive is called to measure the desired variables. Furthermore, an intrusion detection system (IDS) can also be active to detect malicious nodes. In this case, the *delete* primitive can be used to delete a virtual network element/link or even an entire network if an attack (e.g., Distributed Denial of Service - DDoS) is detected.

2.2 Network Virtualization Approaches

Network virtualization platforms must provide the above mentioned functionalities. We now evaluate the main approaches for creating these models according to the level at which the network virtualization is placed.

Fig. 3 compares two basic approaches for virtualizing a network element. Fig. 3(a) shows the conventional network element architecture, with a single control and data plane. For a router, the control plane is responsible of running the network control software, such as routing algorithms (e.g., RIP, OSPF, and BGP) and network control protocols (e.g., ICMP), whereas the data plane is where forwarding tables and hardware data paths are implemented. To virtualize the routing procedure means that a virtualization layer is placed at some level of the network element architecture in order to allow the coexistence of multiple virtual network elements over a single physical network element. Assuming a virtualization layer placed between the control and data planes, then only the control plane is virtualized, as shown in Fig. 3(b). In this case, the data plane is shared by all virtual networks and each virtual network runs its own control software. Compared to the conventional network architecture, this approach greatly improves the network programmability because now it is possible to run multiple and customized protocol stacks, instead of a single and fixed protocol stack. For instance, it is possible to program a protocol stack for network 1, which is different from networks 2 and 3, as illustrated in the figure. In the second network virtualization approach, both control and data planes are virtualized (Fig. 3(c)). In this case, each virtual network element implements its own data plane, besides the control plane, improving even more the network programmability. This approach allows customizing data planes at the cost of some performance loss, because the data plane is no longer dedicated to a common task. This tradeoff between network programmability and performance is investigated in detail in Sections 4.1 and 4.2.



(a) Conventional architecture or monist model: only one network.

(b) Pluralist model virtualizing the control plane (CP).

(c) Pluralist model virtualizing the control plane (CP) and the data plane (DP).

Fig. 3 Approaches for network virtualization differ on the level at which the virtualization layer is placed: a) no virtualization in the monist model; b) pluralist model with several virtual networks with the same data plane, but differing in control plane, and c) pluralist model with several virtual networks differing in control and data planes.

It is worth mentioning that the approach that virtualizes only the control plane, can be further divided into more subcategories depending on the isolation level in data plane sharing among virtual network elements. If there is a strong isolation, then each virtual control plane has access to only a part of the data plane and cannot interfere with the other parts. On the other hand, if the data plane is really shared among virtual control planes, then it is possible that a virtual control plane interferes with other virtual control planes. For instance, it is possible that a single virtual control plane fills the entire forwarding table with its own entries, which can lead to packet drops on the other virtual networks. The decision between strong isolation (slicing) and weak isolation (sharing) is analogous to the decision between circuit and packet switching.

3 Network Virtualization Technologies

In this section, we present two technologies that can be used to network virtualization: Xen and OpenFlow.

3.1 Xen

Xen is an open-source virtual machine monitor (VMM), also called hypervisor, that runs on commodity hardware platforms [7]. Xen architecture is composed of one virtual machine monitor (VMM) located above the physical hardware and several domains running simultaneously above the hypervisor, called virtual machines, as shown in Fig. 4. Each virtual machine has its own operating system and applications. The VMM controls the access of the multiple domains to the hardware and also manages the resources shared by these domains. Hence, virtual machines are isolated from each other, i.e., the execution of one virtual machine does not affect the performance of another. In addition, all the device drivers are kept in an isolated driver domain, called *domain 0* (dom0), in order to provide reliable and efficient hardware support [7]. Domain 0 has special privileges compared with the other domains, referred to as user domains (domUs), because it has total access to the hardware of the physical machine. On the other hand, user domains have virtual drivers that communicate with dom0 to access the physical hardware.

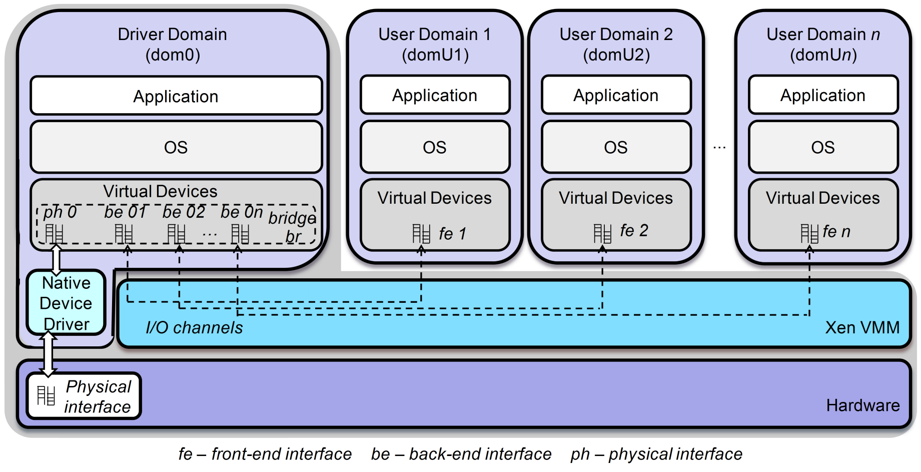
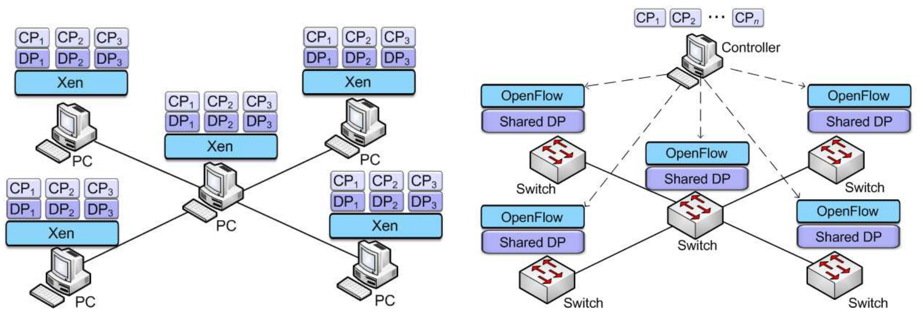


Fig. 4 The Xen architecture.

Xen virtualizes a single physical network interface by demultiplexing incoming packets from the physical interface to the user domains and, conversely, multiplexing outgoing packets generated by these user domains. This procedure, called network I/O virtualization, works as follows. Domain 0 directly access I/O devices by using its native device drivers and also performs I/O operations on behalf of domUs. On the other hand, user domains employ virtual I/O devices, controlled by virtual drivers, to request dom0 for device access [13], as illustrated in Fig. 4. Each user domain has virtual network interfaces, called front-end interfaces, required by this domain for all its network communications. Back-end interfaces are created in domain 0 corresponding to each front-end interface in a user domain. The back-end interfaces act as the proxy for the virtual interfaces in dom0. The front-end and back-end interfaces are connected to each other through an I/O channel. In order to exchange packets between the back-end and the front-end interfaces, the I/O channel employs a zero-copy mechanism that remaps the physical page containing the packet into the target domain [13]. It is worth mentioning that as perceived by the operating systems running on the user domains, the front-end interfaces are the real ones. All the back-end interfaces in dom0 are connected to the physical interface and also to each other through a virtual network bridge. This is the default architecture used by Xen and it is called bridge mode. Thus, both the I/O channel and the network bridge establish a communication path between the virtual interfaces created in user domains and the physical interface.

Different virtual network elements can be implemented using Xen as it allows multiple virtual machines running simultaneously on the same hardware [7], as shown in Fig. 5(a). In this case, each virtual machine runs a virtual router. Because the virtualization layer is at a low level, each virtual router can have its own control and data planes. The primitives for virtual networks, defined in Section 2.1, are easily enforced by using Xen to build virtual routers. First, the execution of one virtual router does not affect the performance of another one because user domains are isolated with Xen. In addition, virtual routers are instantiated, configured, monitored, and deleted on demand. Finally, the live-migration mechanism implemented by Xen [5] allows virtual routers to move over different physical routers.



(a) Xen: one data plane (DP) and one control plane (CP) per virtual router.

(b) OpenFlow: a shared data plane (DP) per node and all the control planes (CPs) on the controller.

Fig. 5 Virtual networks with Xen and OpenFlow.

3.2 OpenFlow

OpenFlow [12] allows the use of the wiring closets on university campus not only for the production network, but also for experimental networks. The OpenFlow project, proposed by Standford University, aims at creating virtual environments for innovations in parallel with the production network using network elements such as switches, routers, access points, and personal computers.

OpenFlow presents a new architecture for providing virtual network environments. The key idea is the physical separation of the network forwarding function, carried out by the data plane, and the network control function, carried out by the control plane. Hence, data and control planes are performed by different network elements. The virtualization of the forwarding elements is accomplished by a shared flow table, which represents the data plane, and all control planes are centralized in a network element, called controller, which runs applications that control each virtual network. An example of network using OpenFlow is on Fig. 5(b).

The OpenFlow protocol defines the communication between forwarding elements and the controller. It is based on the establishment of a secure channel between each forwarding element and the controller, which uses this channel to monitor and configure the forwarding elements. Basically, OpenFlow defines a flow as a sequence of packets and performs forwarding based on flows. Every time the first packet of a not yet classified flow reaches a forwarding element, it is forwarded to the controller. Then, the controller sets a path for the following packets of the flow by setting forwarding rules in each forwarding element that belongs to the chosen path. The controller may also set the action of normal processing for a flow to be forwarded according to conventional layer-2 (L2) and layer-3 (L3) routing, as if OpenFlow did not exist. That is the reason why OpenFlow can be used in parallel to the production network without affecting production traffic.

The data plane in OpenFlow is a flow table described by header fields, counters, and actions. The header fields are a twelve-tuple structure that describes the packet header, as shown in Fig. 6. These fields specify a flow by setting a value for each field or by using a wildcard to set only a subset of fields. The flow table also supports the use of subnet masks, if the hardware in use also supports this kind of match [15]. This twelve-tuple structure gives high flexibility for the forwarding function, because a flow can be forwarded based not only on the destination IP, as in the conventional TCP/IP network, but also on the TCP port, the MAC address, etc. Because the flows can be set based on layer-2 addresses, the forwarding elements of OpenFlow are also called OpenFlow switches. This, however, does not imply that forwarding in OpenFlow must be based on layer 2. Moreover, one of the future objectives of OpenFlow is that the header fields are user-described, which means that the packet header will not be described by fixed fields in a flow, but by a combination of fields specified by the administrator of the virtual network. Thus, OpenFlow will be able to forward flows belonging to networks with any kind of protocol stack.

After the header fields, the flow description is followed by the counters, which are used for monitoring forwarding elements. Counters compute data such as the flow duration and the amount of bytes that were forwarded by one element. The last fields in the flow description are the actions, which are a set of instructions that can be taken over each packet of a specific flow in the forwarding elements. These actions include not only forwarding a packet to a port, but also changing header fields such as VLAN ID, priority, and source/destination addresses.

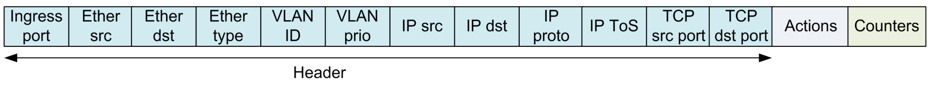


Fig. 6 A flow entry in an OpenFlow forwarding element.

The controller is a central element in the network, which communicates with all the nodes to configure the flow tables. The controller runs a network operating system, which provides the basic functions of network configuration to the applications that manage the virtual networks. Hence, the controller in OpenFlow works as an interface between the network applications and the forwarding elements, providing the basic functions for accessing the first packet in flows and for monitoring network elements. OpenFlow works with any controller that is compatible with the OpenFlow protocol, such as Nox [10]. In this case, each control plane is composed of a set of applications running over Nox. Hence, a virtual network in OpenFlow is defined by its control plane and by the flows that are being controlled by this control plane, as shown in Fig. 7. Hence, the virtual network topology depends on the current flows in the network.

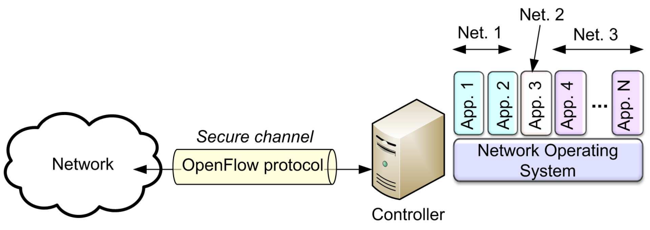


Fig. 7 The OpenFlow controller model.

Using the single controller model, it is possible to create many virtual networks. It is important noticing, however, that different applications running over the same operating system are not isolated. As a consequence, if one application uses all the available resources or crashes it can stop the controller, harming all the other virtual networks. FlowVisor is a tool used with OpenFlow to allow different controllers working over the same physical network [16]. FlowVisor works as a proxy between the forwarding elements and the controller, assuming, for instance, one controller per network. Using this model, it is possible to guarantee that failures in one virtual network will not impact the other virtual networks.

OpenFlow provides a flexible infrastructure based on the idea of distributed forwarding elements, which provide basic functions for operating a network, and centralized control planes. Using this infrastructure, it is possible to slice the physical network into multiple virtual networks. In OpenFlow, the instantiation of a network is just the creation of sets of applications in the controller. The new network flows will be created on demand, according to the packets that enter the network. OpenFlow also provides a flexible infrastructure for reallocating network resources, which means only to reprogram the flow table in each element of the network. This is a simple operation for the controller, because it knows where the physical elements are and how they are connected.

4 Characteristics of Xen and OpenFlow Virtualization Technologies

Neither Xen nor OpenFlow were developed for supporting a pluralist architecture for Internet, but they are the best commodity alternatives for a virtual network substrate. We evaluate the main characteristics of each of these technologies, emphasizing the advantages and the disadvantages for supporting multiple networks and providing flexibility for innovations.

Xen and OpenFlow have different virtualization concepts. Xen creates virtual networks by slicing physical network elements into different concurrent virtual routers. Consequently, a virtual network is seen as a set of interconnected virtual routers distributed over the physical infrastructure. Differently, OpenFlow creates virtual networks by slicing the network control into many control planes, which create the forwarding tables in each network element. Hence, when using OpenFlow, a virtual network is a set of flows with common characteristics, which are controlled by the same set of applications of the OpenFlow controller. The differences between Xen and OpenFlow virtualization models impact scalability, programmability, and network-data processing/forwarding.

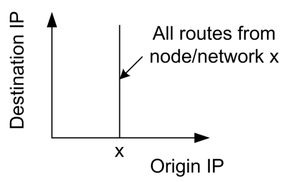
4.1 Network-data Processing and Programmability

One of the main advantages of the pluralist model is to support innovation. As a consequence, the network must be flexible enough providing end-to-end paths over the available physical infrastructure, guaranteeing to the administrator the whole control of the network, which includes, the choice of the protocol stack, the forwarding rules, the network-data processing, etc.

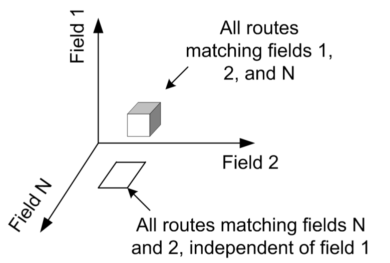
Because Xen virtualization layer is directly placed over the hardware, each virtual router has access to all computer components, such as memory, processor, and I/O devices. Therefore, the network administrator is free to choose everything that runs over the virtualization layer of Xen. Thus, different operating systems, forwarding tables, forwarding rules and so on, can be defined for each virtual network. Furthermore, both data and control plane can be completely virtualized, as shown in Fig. 3(c). Therefore, Xen provides a powerful and flexible platform for the network control and management, allowing hop-by-hop packet processing and forwarding. This way, virtual networks with new functionalities can be easily deployed. For instance, a virtual network with support for packet signature can be instantiated to guarantee authentication and access control. This functionality would solve security problems of the current Internet that cannot be implemented due to the network "ossification" [6]. Even disruptive network models can be implemented due to Xen flexibility for packet processing.

The OpenFlow virtualization model is different from Xen, because the virtual slice is a set of flows and, as a consequence, the actions concern flows, instead of packets. OpenFlow provides a simple packet forwarding scheme in which the network element looks for a packet entry on the flow table to forward the packet. If there is no entry, the packet is forwarded to the controller that sets a forwarding rule in each node on the selected route to forward the packet. Hence, the main disadvantage of the OpenFlow is that all virtual networks have the same forwarding primitives (flow table lookup, wildcard matching, and actions), because the data plane is shared by all the virtual networks in each network node.This, however, does not imply in a completely inflexible packet processing. Indeed, OpenFlow protocol version 1 specifies that the controller can set flow actions that define that a header field can be modified before forwarding the packet. Hence, OpenFlow provides a fine grained forwarding table, much more flexible than the current TCP/IP forwarding table. For instance, the forwarding element could change the destination address to forward the packet to a middle box before forwarding it to the next network element. On the other hand, packet-level features, such as packet signature verification, are not easily implemented in OpenFlow because such features must be executed by the controller or by a middle box, which causes a great loss in the network performance.

Opposing to the OpenFlow shared-data-plane model, Xen provides independent data planes for different virtual networks. Neverthless, Xen is still based on the current TCP/IP forwarding table. Currently, Xen provides a forwarding table which is based on IP routing, which means that the forwarding plane is only based on the source and destination IP addresses. In contrast, OpenFlow flow space definition is composed of n dimensions, where n is the number of fields in the header that could be used to specify a flow, as shown on Fig. 8. Hence, we define a flow based on all the dimensions or based on a wildcard that defines which header fields are important for forwarding packets of that flow [12]. The consequence of this kind of forwarding table is that the packets are forwarded based not only on the destination IP, but also on other parameters, such as the kind of application that is in use. This kind of forwarding table is also possible in Xen, but it is still not available.



(a) Flow space definition in the TCP/IP model.


Field 1
All routes matching fields 1, 2, and N
Field 2
Field N
All routes matching fields N and 2, independent of field 1

(b) Flow space definition in the OpenFlow model.

Fig. 8 Models of flow space to define the forwarding table in TCP/IP based networks and in OpenFlow based networks.

Another key difference between Xen and OpenFlow in what concerns programmability is the control plane model. In Xen, each virtual network node has both the data and the control plane, and, consequently, the network control is decentralized. In OpenFlow, the network node has only the data plane. The control plane is centralized on the controller, which is a special node in the network. The use of a centralized control plane makes it easier to develop algorithms for network control, when compared to the use of a decentralized approach. A centralized control, however, creates the need for an extra server in the network and also creates a single failure point in the network.

4.2 Performance on Network-data Forwarding

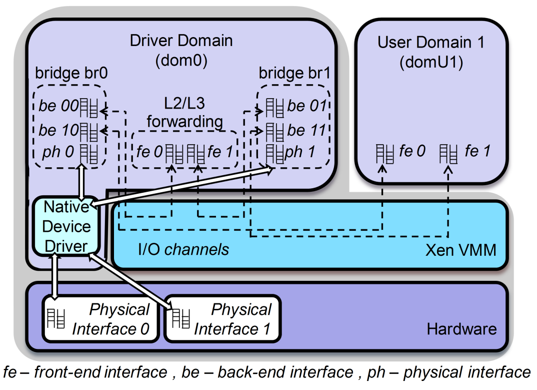
One important feature of a technology to provide an environment with multiple virtual networks for the Future Internet is a high forwarding performance. The efficiency of network-data forwarding does not uniquely depend on the hardware, but also on the logic provided by each technology. In this section, we assume that both Xen and OpenFlow run in the same hardware to evaluate which losses each technology imposes to the network-data forwarding.

Xen performance depends on the location where packet forwarding is performed. For each virtual router, packet forwarding can be performed by the operating system running on the user domain corresponding to the virtual router or by domain 0. In the first case, we have more flexibility in the packet processing, but the costs associated with moving packets between dom0 and domU, to perform forwarding, introduces control overhead and impact Xen performance. In the second case, packets to and from all virtual routers are forwarded by dom0, which deals with multiple forwarding tables simultaneously.

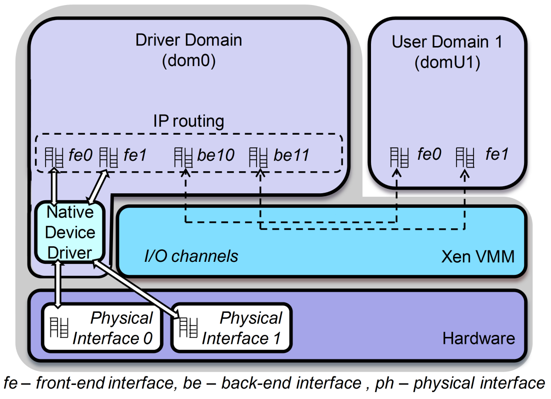
The Xen performance of packet forwarding also depends on the two possible modes used to move packets among virtual machines [7]: bridge and router modes. The bridge mode is the default network architecture used by Xen, presented in Fig. 4. Nevertheless, this architecture does not apply for a router, because we need more than one physical interface in each device. Fig. 9(a) shows an example of the bridge mode with two physical interfaces. We have two bridges on dom0, one per physical interface, connecting the back-end interfaces and the physical ones. Packet forwarding, in this case, can be performed at dom0 by using layer-2 or layer-3 forwarding. Let p be a packet arriving at physical interface $ph0$ that must be forwarded to physical interface $ph1$. First, p is handled by the device driver running on dom0. At this time, p is in $ph0$, which is connected to bridge $br0$. This bridge demultiplexes the packet p and moves it to back-end interface $be00$ based on the MAC address of the frame destination. After that, p is moved from $be00$ to the front-end interface $fe0$ by using the I/O channel through the hypervisor. The packet p is then forwarded to the front-end interface $fe1$ and after that another I/O channel is used to move p to the back-end interface $be01$. This interface is in the same bridge $br1$ of the physical interface $ph1$. Thus, p reaches its outgoing interface. It is worth mentioning that the hypervisor is called twice to forward one packet.

In the router mode, illustrated by Fig.9(b), the domain 0' interfaces are the physical ones with an IP address associated to each one. As a consequence, the router mode does not require bridges connecting each physical interfaces and I/O channels, i.e., packet forwarding from a physical interface to another one at dom0 is performed as well as in native Linux. In this case, if domain 0 is used as shared data plane (Fig. 3(b)), there are no calls to the hypervisor. With the router mode, the hypervisor is called only when each virtual router implements its own data plane, as illustrated in Fig. 3(c). In this case, packets are routed to the back-end interface associated to the destination domU and then are moved to the front-end interface by using the I/O channel through the hypervisor. Then, packets are moved to the back-end interface and finally routed to the outgoing physical interface. In order to allow user domains to send and receive packets, IP addresses are also assigned to back-end interfaces in contrast to the bridge mode.

OpenFlow does not assume virtualized data planes on forwarding elements and, consequently, follows the model of one data plane for all the networks. Consequently,



(a) The bridge mode.



(b) The router mode.

Fig. 9 The Xen network architectures for packet forwarding.

it is expected for OpenFlow performance the same performance of the native packet forwarding. OpenFlow, however, shows a disadvantage when the flow is not configured. As we explained before, when a packet reaches an OpenFlow switch, if the flow is not configured on the table, it is forwarded through the network to the controller. The controller, then, configures the OpenFlow switches to route the packet through the network. This mechanism introduces a greater delay when forwarding the first packet of each flow, due to the transmission and the controller processing delays. If the traffic is mostly formed of small flows, it can imply in a performance decrease in OpenFlow.

4.3 Scalability to the Number of Virtual Networks

Scalability is related to the number of virtual networks running over the same physical node. The new Internet requisites are still an open issue and the new architecture should not restrict the number of networks running over the available physical infrastructure. The Xen approach is less flexible in this sense, because the virtual network element is a virtual machine, which demands much more hardware resources, such as processing power and memory space, than a simple set of flows in an OpenFlow switch. Indeed, context switching and datapath in Xen are much more complex than in OpenFlow. The concept of virtual networks in OpenFlow is given by a set of flows which corresponds to a specific set of characteristics that define that virtual network. For this reason, OpenFlow supports thousands of virtual networks running in parallel, while Xen is restricted to the number of virtual machines that can be multiplexed over the same hardware. It is worth mentioning that Xen scalability can be improved if Domain 0 is used as a shared data plane.

5 Performance Evaluation

We evaluate the performance of Xen and OpenFlow in a testbed composed of three machines, as shown in Fig 10. The Traffic Generator machine (TG) sends packets to the Traffic Receiver machine (TR), through the Traffic Forwarder machine (TF), which simulates a virtual network element. The Traffic Forwarder machine (TF) is an HP Proliant DL380 G5 server equipped with two Intel Xeon E5440 2.83GHz processors and 10GB of RAM. Each processor has 4 cores, therefore TF machine can run 8 logical CPUs. When not mentioned, TF machine is set up with 1 logical CPU. TF machine uses the two network interfaces of a PCI-Express x4 Intel Gigabit ET Dual Port Server Adapter. The Traffic Generator and Traffic Receiver are both general-purpose machines equipped with an Intel DP55KG motherboard and an Intel Core I7 860 2.80GHz processor. Traffic Generator (TG) and Traffic Receiver (TR) are directly connected to the Traffic Forwarder (TF) via their on-board Intel PRO/1000 PCI-Express network interface.

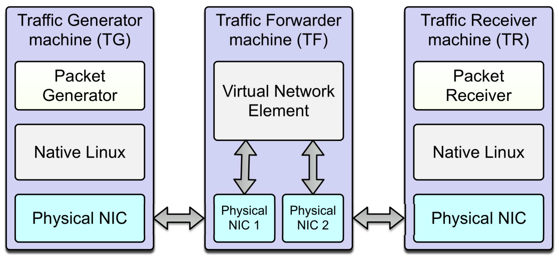


Fig. 10 Testbed used in the evaluation. The Traffic Forwarder machine (TF) is set as Xen, OpenFlow, or Native Linux, according to each experiment.

In the following experiments, we test packet forwarding using Native Linux, Xen, and OpenFlow.

In Native Linux experiments, the Traffic Forwarder runs a Debian Linux kernel version 2.6.26. This kernel is also used in OpenFlow experiments with an additional kernel module to enable OpenFlow. In Xen experiments, Domain 0 and User Domains run a Debian Linux system with a paravirtualized kernel version 2.6.26. For traffic generation, we use the Linux Kernel Packet Generator [14], which works as a kernel module and can generate packets at high rates. In the following, we explain the packet forwarding solutions evaluated in our experiments.

5.1 Xen, OpenFlow, and Native Linux Scenarios

In the Xen scenario, we test three different network configurations. In the two first ones, Xen works in the bridge mode, explained in section 4.2. In the first configuration, called Xen-VM, virtual machines work as complete virtual routers, which means that both data and control plane are on the virtual machine. In the second configuration, called Xen-Bridge, we assume that virtual machines contain only the control plane. The data plane, running in Domain 0, is shared by all virtual routers. The Xen-Bridge configuration is expected to give a higher performance on packet forwarding, but it reduces the flexibility on packet processing when compared with the Xen-VM configuration. Finally, in the third configuration, Xen works in the router mode. In this case, we evaluate only the packet forwarding through Domain 0 and we call this configuration Xen-Router. We assume, for this configuration, the existence of a forwarding table in Domain 0 corresponding for each virtual machine. We use the Xen hypervisor version 3.4.2 for all configurations.

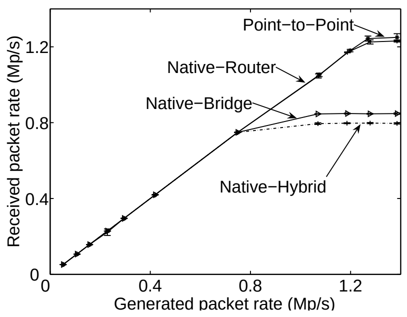
In the OpenFlow scenario the Traffic Forwarder (TF) acts as an OpenFlow Switch. An OpenFlow Controller is connected to TF, using a third network interface. TF runs OpenFlow Reference System version 0.8.9. The controller is an IBM T42 Laptop that runs a Debian Linux system. We choose Nox version 0.6.0 [10] as the network controller. We use the `pyswitch` application, which is available in Nox to create flow rules in the OpenFlow switch.

In the Native Linux scenario, we test three different packet forwarding configurations. In the first one, Native-Router, TF works as a router. For this test we used the standard Linux kernel routing mechanism with static routes. The Native-Bridge configuration uses the Linux kernel bridge, which implements a software-based switch on the PC. Since we compare layer-2 and layer-3 solutions with OpenFlow and Xen, we need to compare their performance with both bridge and router modes of native Linux to evaluate the impact of virtualization on packet forwarding. Xen in the bridge mode, however, has a different configuration from the native Linux with bridge. This is because Linux bridge does L2 forwarding between two physical interfaces and Xen goes up to L3 forwarding. To perform a fair comparison between Xen in bridge mode and native Linux, we create an hybrid mode (bridge and router) for native Linux, which we call Native-Hybrid. In this hybrid mode, TF's physical network interfaces are connected to different software bridges and kernel routing mechanism forwards packet between the two bridges. This configuration simulates in native Linux what is done on Xen bridge mode, illustrated in Fig. 9(a).

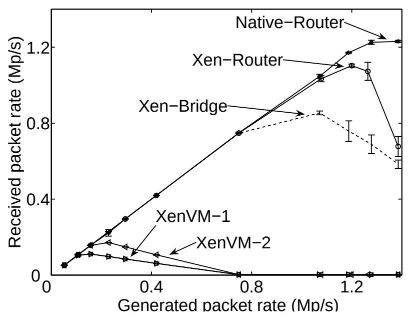
5.2 Experimental Results

Our first experiments measure the forwarding rate achieved by the different packet forwarding solutions. The packet forwarding rate analysis is accomplished with minimum (64 bytes) and large (1512 bytes) frames. We use 64-byte frames to generate high packet rates and force high packet processing in TF and 1512-byte frames to saturate the 1 Gb/s physical link.

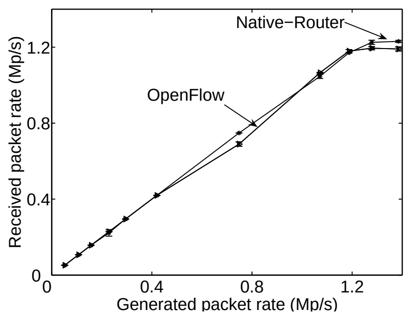
Figs. 11(a) and 11(b) show the forwarding rate obtained with Native Linux, which gives an upper bound for Xen and OpenFlow performances. We also plot the Point-To-Point packet rate, which is achieved when TG and TR are directly connected. Any rate achieved below the Point-To-Point packet rate is caused by loss between TG and TR. The results show that Native Linux in router mode performs as well as the Point-to-Point scenario. This is explained by the low complexity on kernel routing mechanism. In the bridge mode, however, Native Linux performs worse than in router mode. According to Mateo [11] this result may be due to the Linux bridge implementation, which is not optimized to support high packet rates. Finally, we observe that Native Linux in the hybrid mode has the worst forwarding performance. This is an expected result due to the previously mentioned limitations of bridge mode and the incremental cost required to forward packets from the bridge to IP layer in TF.



(a) Native Linux.



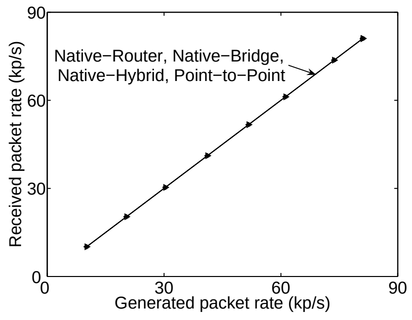
(b) Xen.



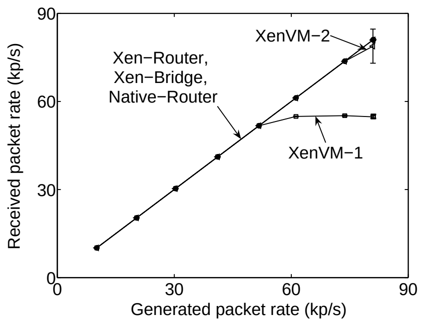
(c) OpenFlow.

Fig. 11 Packet rate for different forwarding elements, using 64-byte frames.

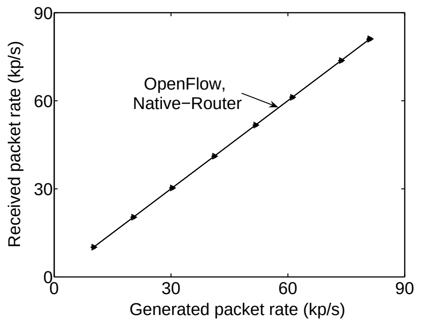
The forwarding rate results for Xen are shown in Fig. 11(b). First, we analyze a scenario where Domain 0 forwards the packets. In this scenario no virtual machine is running, although the same results are expected when virtual machines are up and they do not forward packets [7]. In this experiment, we test the Xen bridge and router modes. Xen-Bridge uses the Linux bridge to interconnect the virtual machines, as explained in Section 4.2. Xen-Bridge suffers the same limitations of Native Linux in bridge mode, since the bridge implementation is the same. In addition, Xen-Bridge forwards packets



(a) Native Linux.



(b) Xen.



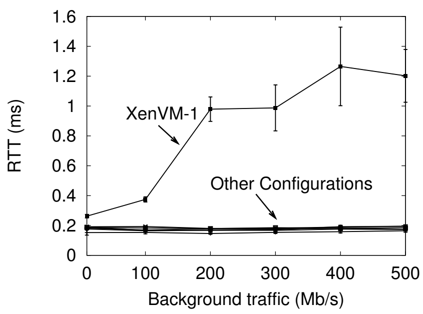
(c) OpenFlow.

Fig. 12 Packet rate for different forwarding elements, using 1512-byte frames.

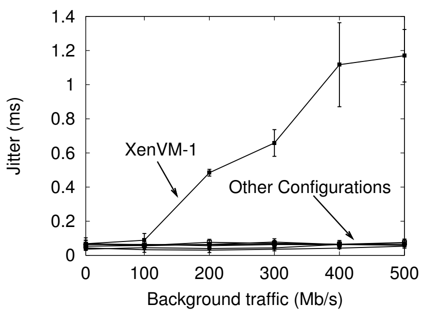
from the bridge to IP layer, as in hybrid mode, combined with hypervisor calls necessary in this mode. As expected, Xen-Bridge performs worse than all Native Linux forwarding schemes. On the other hand, Xen-Router performs better than Xen-Bridge, because the Linux bridge is not used and Xen hypervisor is not called when Domain 0 forwards packets. Nevertheless, Xen-Router is still worse than Native-Router. The forwarding rate rapidly decreases after about 1.2 Mp/s load. This behavior is also observed for Xen-Bridge and in the following experiments with virtual machine forwarding. This performance penalty is related to Xen interrupt handling implementation and needs further investigation. Next, we analyze a scenario where a virtual machine forwards traffic using Xen bridge mode, the default Xen network configuration. In XenVM-1 configuration, both virtual machine and Domain 0 share the same CPU core. This result shows a drop in performance compared with previous results, in which Domain 0 was the forwarding element. At first glance, this poor performance could be caused by high contention for CPU resources due to the fact that a single CPU core is shared between the domains. To eliminate the contention for CPU resources we experiment with XenVM-2 configuration in Fig. 11(b), where we give one exclusive core to each domain. The performance obtained with XenVM-2 experiment is better than with XenVM-1, but it is still lower than Domain 0 results. This can be explained due to the high complexity involving virtual machine packet forwarding. When the traffic is forwarded through the virtual machines, it must undergo a more complex path before reaching TR. Upon packet receiving, it is transferred via DMA to Domain 0 memory. Domain 0 demultiplexes the packet to its destination, gets a free memory page associated with the receiving virtual machine, swaps the free page with the page containing the packet, and then notifies the virtual machine. For a virtual machine to send a packet, it must put a transmission request along with a reference to the memory area where the packet is into Xen I/O ring. Domain 0 then polls the I/O ring and, when it receives the transmission request, it maps the reference into the physical page address, and then sends it to the network interface [4]. This increased complexity is partially responsible for the lower packet rate obtained in the two curves where virtual are used to forward packets.

Fig. 11(c) shows that OpenFlow performs near Native Linux in router mode. In addition, the comparison between OpenFlow and XenVM results shows the tradeoff between flexibility and performance. On XenVM we have more flexibility, because the data and control planes are under total control of each virtual network administrator. In OpenFlow, however, the flexibility is lower because the data plane is shared by all virtual networks. On the other hand, due to lower processing overhead, OpenFlow performs better than XenVM in our scenario. Xen performance can be raised if the data plane is moved to Domain 0, as we can see in Xen-Router and Xen-Bridge results. In this case, however, the flexibility of customizing data planes is decreased.

We also carried out packet forwarding experiments with 1470-byte data packets, shown in Fig. 12. With large packets, all forwarding solutions but XenVM-1 and XenVM-2 have the same behavior as in the Native-Router scenario. It means that there is no packet loss in TF and the bottleneck in this case is the 1 Gb/s link. Nevertheless, with XenVM-1, where a virtual machine shares the same core with Domain 0, the packet rate is achieved is lower. In XenVM-2 experiments, where we give one exclusive CPU core for each domain, the behavior is similar to Native-Router. Thus, we conclude that, in this case, the performance decrease in XenVM-1 result is caused by high contention for CPU resources between domains and giving an exclusive CPU core to Domain 0 solves the problem.



(a) Round trip time (RTT) .

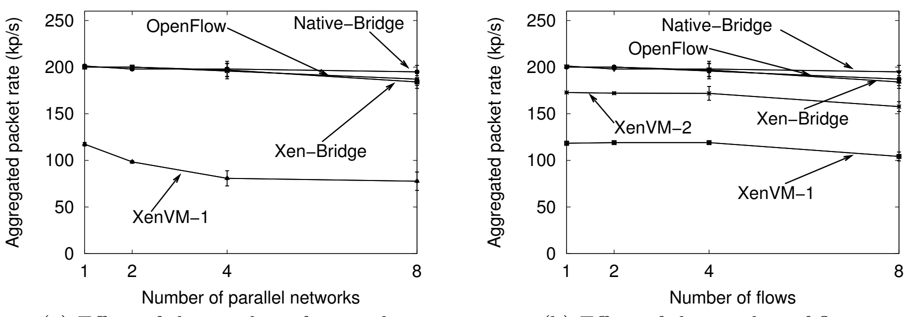


(b) Jitter.

Fig. 13 Analysing network delays according to the network element which forwards the traffic, assuming 128-byte packets.

Next, we analyze how each type of virtual network element impacts the traffic latency. We create background traffic with different rates to be forwarded by the network element. For each of those rates, an ICMP echo request is sent from the generator to the receiver, to evaluate the round trip time (RTT) and the jitter according to the generated background traffic. By analyzing the jitter, defined as the mean of the standard deviation of RTT measures, we investigate if the network element inserts a fixed or a variable delay in the network, which could affect some real-time applications.

Figs. 13(a) and 13(b) show the results for the RTT and the jitter, respectively. As the generated traffic increases, the RTT and jitter of the ICMP messages increase only for the configuration in which the traffic passes through the virtual machine, which we call XenVM-1 in the graph. The difference in the RTT between XenVM-1 and Native-Linux experiments is up to 1.5 ms in the worst scenario, with background traffic of 500 Mb/s. The RTT and the jitter of OpenFlow have the same order of magnitude as the RTT and jitter of Native-Linux. Despite of the delay difference between XenVM-1 and the other configurations, Xen virtual machines can handle network traffic without a significant impact on the latency. Because the RTT is always smaller than 1.7 ms, even in the worst case, virtual routers running over Xen do not significantly impact real-time applications such as voice over IP (VoIP), which tolerates up to 150-ms delay without disrupting the reliability of the communication, even if one considers multiple hops [8].



(a) Effect of the number of networks. (b) Effect of the number of flows.

Fig. 14 Aggregated packet rate according to the number of virtual networks.

We also analyze how each virtualization platform behaves with multiple networks and multiple flows per network. In this scenario, each network is represented as a flow of packets between the TG and TR for OpenFlow, and as a virtual machine for Xen. The packet size and the generated packet rate are fixed in 64 bytes and 200 kp/s, respectively. If there is more than one parallel flow, the aggregated generated traffic is still the same. For example, if the test is performed with four parallel flows, each of them receives a packet rate of 50 kp/s, generating an aggregated rate of 200 kp/s.

Fig. 14(a) shows the aggregated packet rate as a function of the number of virtual networks, with one flow per network. OpenFlow acts like a software switch despite the fact that the first packet of the flow must go to the OpenFlow controller. The performance obtained is very similar to a software bridge running over Native Linux, maintaining the received rate close to the generated rate of 200 kp/s. Although Xen's Domain 0 must have its interrupts first handled by the hypervisor, Xen-Bridge performs almost as well as native Linux in bridge mode. On the other hand, in the case where multiple virtual machines are simultaneously forwarding traffic (XenVM-1 configuration), the performance degrades as the number of parallel virtual machines increases. This is mainly because of the CPU scheduler, which must multiplex the processor among an increasing number of machines, each one requiring to forward its own flow.

Fig. 14(b) shows the aggregated packet rate as a function of the number of flows, considering a single virtual network. As expected, OpenFlow and Xen-Bridge present the same behavior as in Fig. 14(a), because both share the data plane and, consequently, there is no difference between a virtual network with multiple flows and multiple networks with one flow each. On the other hand, when the traffic is forwarded through the virtual machines (XenVM-1 configuration), the traffic must undergo a more complex path before reaching TR, as seen in previous results. In order to verify if the complex path is the only bottleneck, the test was repeated in a configuration where the virtual machine does not share the same physical core with Domain 0, referred to as XenVM-2. In this configuration, the performance is increased by up to 50 kp/s, which indicates that the lack of processor availability is an important issue in network virtualization.

To analyze the impact of CPU allocation on virtual machine forwarding, we have conducted a CPU variation test in which we send packets from TG to TR at a fixed rate of 200 kp/s through virtual machines and vary the number of dedicated CPU cores given to Domain 0. The 200 kp/s rate is used because near this rate we obtain the best performance in the 1-virtual machine scenario. According to previous results, the forwarding performance increases when both Domain 0 and virtual machine have a dedicated CPU core. This test aims to complement those results by analyzing the forwarding performance when the number of Domain 0's exclusive CPU cores increases and more virtual machines forward packets. When more than one virtual machine is used, the global sent rate of 200 kp/s is equally divided among virtual machines. Fig. 15 shows the aggregated received rate in a scenario in which each virtual machine has one single core and the number n of CPU cores dedicated to Domain 0 is varied. According to Fig. 15, the worst performance is obtained when all domains share the same CPU core (i.e., $n = 0$), due to a high contention for CPU resources. As expected, when $n = 1$ the performance increases, because each virtual machine has a dedicated CPU core and, consequently, has more time to execute its tasks. In addition, when Domain 0 receives more than one dedicated CPU core (i.e., $n \geq 2$), the performance is worse than when Domain 0 has a single dedicated CPU core, even when more virtual machines forward packets. These results show that the network tasks that Domain 0 executes when each virtual router has 2 interfaces are single-threaded and these tasks are under-performing in a multi-core environment.

6 Conclusions

In this paper, we investigate the advantages and limitations of Xen and OpenFlow as network virtualization platforms for creating a pluralist architecture for the Future Internet. We show that both Xen and OpenFlow allow the creation of multiple customized virtual networks, but using different approaches. Xen enables a network virtualization model in which the network element is totally virtualized, with both control and data planes residing into a virtual machine. This construction provides a powerful and flexible environment in which the network administrator can program new network protocol stacks and also customized network-data forwarding structures and lookup algorithms. Nevertheless, our experimental results show that this flexibility comes with a performance cost. Our experiments in a personal computer used as

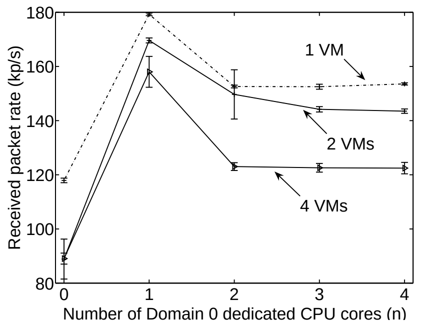


Fig. 15 Received packet rate when varying the number of CPUs allocated to Domain 0.

a software router reveal a high complexity involving Xen virtual machine packet forwarding that limits the forwarding capacity to less than 200 kp/s. We also observe that moving the data planes from virtual machines to privileged domain of Xen avoids hypervisor calls, and thus Xen forwarding capacity improves near to the native Linux performance. In this configuration, however, we lose the flexibility of customizing each virtual router data plane, because shared data planes require the use of the same forwarding mechanisms for all virtual networks. OpenFlow network virtualization model follows the shared data plane approach by defining a centralized element that controls and programs the flow table in each network element. Our results demonstrate that the generalization of a flow to an n-tuple of header fields enables a flexible and yet performant forwarding structure. Our PC-based OpenFlow prototype forwards packets as fast as native Linux. Other experiments show that both Xen and OpenFlow are suitable platforms for network virtualization, because they are proved to support multiple instances of virtual network elements with no measurable performance loss, using the shared data-plane configuration. We will investigate in a future work the benefits of modern hardware-assisted I/O virtualization technologies. We expect that direct access to I/O devices will significantly improve the packet forwarding efficiency through Xen virtual machines.

References

1. T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet impasse through virtualization. *IEEE Computer*, 38(4):34–41, April 2005.
2. P. Baran. On distributed communications networks. *IEEE Transactions on Communications Systems*, 12(1):1–9, March 1964.
3. M. S. Blumenthal and D. D. Clark. Rethinking the design of the Internet: the end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, August 2001.
4. D. Chisnall. *The Definitive Guide to the Xen Hypervisor*. Prentice Hall, 2007.
5. C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Symposium on Networked Systems Design & Implementation - NSDI*, pages 273–286, May 2005.

6. D. Clark, R. Braden, K. Sollins, J. Wroclawski, D. Katabi, J. Kulik, X. Yang, T. Faber, A. Falk, V. Pingali, M. Handley, and N. Chiappa. New Arch: Future generation Internet architecture. Technical report, USC Information Sciences Institute Computer Networks Division, MIT Laboratory for Computer Science and International Computer Science Institute (ICSI), August 2004.
7. N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley. Evaluating Xen for router virtualization. In *International Conference on Computer Communications and Networks - ICCCN*, pages 1256–1261, August 2007.
8. H. Fathi, R. Prasad, and S. Chakraborty. Mobility management for VoIP in 3G systems: Evaluation of low-latency handoff schemes. *IEEE Wireless Communications*, 12(2):96–104, 2005.
9. N. Feamster, L. Gao, and J. Rexford. How to lease the Internet in your spare time. *ACM SIGCOMM Computer Communication Review*, 37(1):61–64, January 2007.
10. N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, July 2008.
11. M. P. Mateo. OpenFlow switching performance. Master's thesis, Politecnico Di Torino, Torino, Italy, July 2009.
12. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S., and J. Turner. OpenFlow: Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, April 2008.
13. A. Menon, A. L. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In *USENIX Annual Technical Conference*, pages 15–28, May 2006.
14. R. Olsson. Pktgen the Linux packet generator. In *Linux symposium*, pages 11–24, July 2005.
15. B. Pfaff, B. Heller, D. Talayco, D. Erickson, G. Gibb, G. Appenzeller, J. Tourrilhes, J. Pettit, K.-K. Yap, M. Casado, M. Kobayashi, N. McKeown, P. Balland, R. Price, R. Sherwood, and Y. Yiakoumis. OpenFlow switch specification version 1.0.0 (wire protocol 0x01). Technical report, Stanford University, December 2009.
16. R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.-Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, S. Seetharaman, D. Underhill, T. Yabe, K.-K. Yap, Y. Yiakoumis, H. Zeng, G. Appenzeller, R. Johari, N. McKeown, and G. Parulkar. Carving research slices out of your production networks with OpenFlow. *ACM SIGCOMM Computer Communication Review*, 40(1):129–130, 2010.
17. 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. In *ACM SIGCOMM*, pages 231–242, August 2008.