Network Virtualization - An Overview

Kilian Rausch
Advisor: Michael Herrmann
Seminar Innovative Internet-Technologies and Mobile Communications
Chair for Network Architectures and Services
Department for Computer Science, Technische Universität München
Email: rauschki@in.tum.de

ABSTRACT

This paper introduces the basic approach of *Network Virtualization*, as well as Xen and OpenFlow as two opportunities to realize this approach. A virtual network is an autonomous, fully isolated network above an existing physical infrastructure. The goal of virtual networks is to run side-by-side to productive networks without affecting them and consequently to enable new network innovations to be tested safely. Xen is a hypervisor running directly on the hardware and able to host a large number of guest systems. This paper presents the qualifications of Xen to act as a virtual router platform. OpenFlow itself is an open standard for network devices to enable the easy deployment of experimental networks over an existing infrastructure. The related FlowVisor project expands OpenFlow by the opportunity to create and administrate fully isolated virtual networks. These two approaches were chosen among others because of their advanced progress and relevancy to practice.

Keywords

Network Virtualization, Router Virtualization, Software Designed Networking, Virtual Networks, Xen, OpenFlow, FlowVisor

1. INTRODUCTION

In the past years a high interest in reconsidering the existing network and Internet architecture came up. Some even describe the todays Internet architecture as "ossified" [10]. This movement was mainly carried by researchers, wanting to experiment with new network innovations. But building a realistic network environment for experimental purposes would require enormous investments. Therefore the success of new protocols and network architectures depends the possibility to run and test them coexistent, but isolated to existing network infrastructures. So they can not affect the productive networks, but can be tested in detail under real conditions. Productive networks are networks, which reliably carry out the everyday load. *Network Virtualization* is an efficient way to overcome this obstacle and to pave the way for new network ideas and developments. Network Virtualization is generally achieved through running an additional software on the network devices.

Besides the named above, Network Virtualization offers other various benefits. For example it allows the network operators to save a fix state of the network (In this paper we will describe this operation with the term *"Checkpoint Saving"*). This works like saving an image of a virtual operating system (OS) and could be very useful before changes are deployed. Then the previous state of the complete virtual network can be restored in the case of an error.

Another application scenario would be, that Network Infrastructure Providers have the potential to host multiple customers on the same hardware. Consequently they could coordinate the load in a much more efficient way. This leads to immense cost reductions for both parties and promotes competition, because then even smaller network operators have the possibility to use large-scale network infrastructure. Recently the *Open Network Foundation* (ONF) was established to support the development of the so called *Software Designed Networking* (SDN). Among the members are leading players like Facebook, Google, Microsoft, Deutsche Telekom and IBM. This non-profit organization especially supports the development and rollout of OpenFlow.

This paper gives an overview of Network Virtualization and presents two possible solutions and their relevancy for praxis. On that account we will especially look at the problems and challenges, which were discovered in several test cases. Given, that Virtualization in general brings a performance loss, we will highlight this topic in the dedicated Sections. In Section 2 the general approach of Network Virtualization is illustrated. In Section 3 Xen Hypervisor as solution for hosting multiple virtual routers is presented and in Section 4 we deal with OpenFlow and FlowVisor as the second possible solution. Finally we conclude in Section 5.

2. NETWORK VIRTUALIZATION - OVERVIEW AND GENERAL APPROACH

Virtualization in general is a method to concentrate or divide resources of a computer. It abstracts from the physical hardware, but gives the user the impression of interacting directly with it through the allocation of hardware resources. Besides the rapid development of virtualization of operating systems, researchers began to pay more attention to the virtualization of routers. This means running several virtual routers on the same machine as the basis of administrating multiple networks. Every router then belongs to a dedicated network, giving the full administrative power. The approach of Network Virtualization itself is not really new, as we already know *Virtual Private Networks* (VPNs) as solution for connecting different networks. The difference between VPNs and Virtual Networks (VNs) is illustrated in [1, page 2] as follows:

Although VPNs provide a virtualized channel above a physical network infrastructure, they have a few disadvantages in comparison with Virtual Networks. So all virtual networks have to be based on the same protocols, topology and

doi: 10.2313/NET-2011-07-2_12

addressing schemes. This mainly penalizes the reasearchers, wanting to deploy many different kinds of experimental networks. Another big issue is the missing isolation of the VPNs. While a few workarounds exist to deal with that problem, even these solutions provide no real isolation. Furthermore the infrastructure provider and the VPN service provider are normally the same entities. This disadvantage is among other things caused by the missing resource isolation. So multiple providers can not be sure, that another one is not affecting his network (for example by stressing the infrastructure). But the most important advantage of Virtual Networks over VPNs is the true isolation of the different Virtual Networks [1, page 2]. This mainly targets on hiding of network infrastructure specifications and even the existence of another administrative domain. But of course, we must consider the different use cases of VPNs and VNs. VPNs are often used to safely connect two networks or to establish a single connection remotely, for example to and within company networks. On the other hand a Virtual Network is used when researchers want to deploy a complete network over an existing one to test it under real conditions. So the Virtual Network provides the feature to be programmed individually with experimental protocols, address schemes et cetera.

Another commonly used technique are Virtual Local Area Networks (VLANs).VLANs are able to set virtual links above physical ports. The difference to Virtual Networks is, that they only can virtualize one specific forwarding algorithm. So the flexibility of choosing and developing new network innovations is not supported here [12].

The basic elements of a virtualized network environment are the *substrate layer*, where the physical resources (like CPU, memory or storage) are located, and the *virtual network layer*. The substrate layer contains the *substrate nodes*, mostly represented through typical network hardware like routers or switches, which must support virtualization. On these substrate nodes multiple virtual nodes are hosted, as well as the physical link contains multiple virtual links. Figure 1 shows the interaction of the virtual and the substrate layer. Virtual links are bundled to an aggregate. In order to clearly identify a specific VN in a wide infrastructure, a Virtual Network ID is used. This ID is globally unique and consists of the responsible organization ID and the virtual link ID [1]. When a Virtual Network is set up, a special *VNP/InP Inteface* provides all relevant information needed to build the VN, like the topology of nodes and virtual links, physical location and traffic characteristics.

3. XEN

In this Section Xen and an approach to use its virtualization technology to build and operate a virtual network is introduced. Xen is a hypervisor on x86 base which allows multiple operating systems to run on the same hardware. It is widely known as virtualization platform for operating systems. Xen is operating directly on the hardware and is able to host different OSs in the so called domains at the same time. The primary intention was to keep the performance overhead as small as possible to simultaneously host up to 100 Virtual Machines [3, page 1] and to isolate the domains safely. In the following we will introduce how the virtualization capabilities of Xen can be used to host virtual

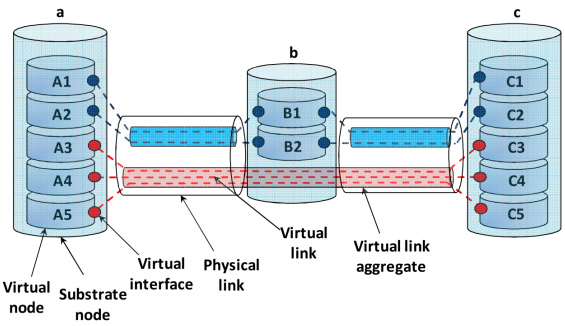

Figure 1: The substrate layer contains the *substrate nodes*. On these substrate nodes multiple virtual nodes are hosted. One physical link contains multiple virtual links, mostly bundled to an aggregate. [1]

routers.
Mainly the following challenges occurred when developing Xen [3, page 1]:

1. True Isolation of the Virtual Machines
2. Support of various operating systems
3. Keep the performance overhead as small as possible

3.1 Overview and Design Issues

In a traditional *Virtual Machine Monitor* (VMM) the hardware is emulated [5]. In contrast to this technique Xen uses paravirtualization to host a guest OS. This means a software interface is provided for the hardware; similar, but not equal to the hardware (no hardware emulation). As a consequence of that some slight modifications of the OS are indispensable. But the relative low costs to port an OS to Xen are worth to invest. This results in a system, which is nearly as efficient as a native system [13]. Figure 2 shows the efforts, measured in lines of codes (LOC), of porting a convenient OS to Xen.

OS subsection	**# lines**	
	Linux	**XP**
Architecture-independent	78	1299
Virtual network driver	484	–
Virtual block-device driver	1070	–
Xen-specific (non-driver)	1363	3321
Total	**2995**	**4620**
(Portion of total x86 code base	**1.36%**	**0.04%)**

Figure 2: porting costs, measured in lines of codes (LOC), of porting Linux or Windows®XP to Xen. [3]

Important to mention is, that the the guest applications remain unaffected here. The big difference of Xen to a convenient operating system, hosting the virtual machines ("normal" Virtualization), is the possibility to provide *performance isolation* [4]. Performance isolation ensures, that

doi: 10.2313/NET-2011-07-2_12

one virtual machine's performance can not impact the performance of another one (what of course is a desirable goal). But as in [4, page 19] mentioned, this works only under certain conditions. We will treat this especially for virtual router purposes in detail in the Section "Evaluation and Performance".

3.2 Xen and the x86 architecture

The Virtual Machine Interface for x86 architecture consists of three main elements [3]. *Memory Management, CPU Management* and *I/O Device Management*.

3.2.1 Memory Management

The memory management on x86 is quite difficult, because there is no software-managed *Translation Lookaside Buffer* (TLB), which translates virtual memory to physical memory. So Xen is designed to hand over the responsibility of allocating and managing the hardware page tables to the guest OSs.

3.2.2 CPU Management

In CPU Management x86 can play to its strength by having four *privilege levels* (alternatively called *rings*). Most other architectures have only two privilege levels. Mostly one privilege level for sensitive and one for non-sensitive commands (Popek and Goldberg Theorem [7]). Because Xen must urgently be located in level 0 (to have all possible rights), here the problem occurs, that the guest system has to share the same level with its applications. The guest system then runs in a separate address space. This leads to expensive TLB flushes through permanently involving the hypervisor for access control of the applications [3, page 4]. Running a guest system in level 0 would completely destroy the idea of isolation, because then this system would be in the position to change the whole system and consequently the other guest systems too. So the four rings of x86 are essentially needed to assign the highest privilege level to Xen, the second one to the guest OS and the third or fourth one to the applications. This avoids any influence of errors to the hypervisor and the expensive TLB Flushes.

3.2.3 I/O Device Management

The complete I/O Device Communication of each domain is processed by Xen with the embedded interfaces of several device abstractions. As mentioned above, the hardware is not emulated. The embedded interfaces in combination with the adapted OS result in much more performance. Additionally this allows Xen to manage and ensure the isolation of the guest operating systems.

In Figure 3 a sample configuration with Xen and different operating systems is illustrated. Dom0 must be emphasized as privileged control domain of Xen, able to start and stop other domains (to make this possible, this functionality should be implemented in the operating system of dom0. Here a XenoLinux is used).

3.3 Evaluation and Performance

As mentioned before in Section 1 there is a high interest of dedicated network operators and researchers in changing the existing network and Internet architecture. This Section

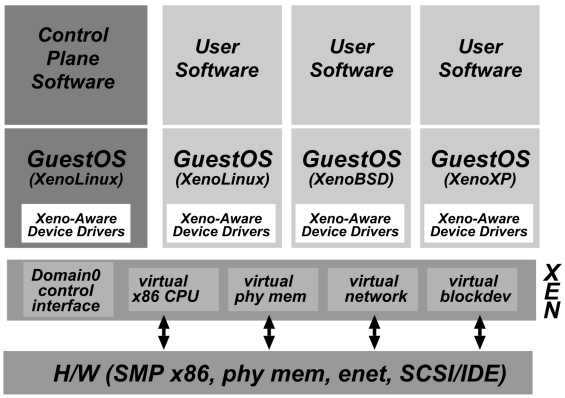

Figure 3: Architecture of a machine using Xen with different guest operating systems and the privileged dom0 control domain [3]

treats the capability of Xen to be used as a system for driving virtual routers. Multiple logically independent software routers are hosted on one single hardware. Each of them is responsible for his own isolated Virtual Network. The following statements and evaluations are based on [6] and [2]. Enabled by by powerful hardware and virtualization support, Xen's guest domains (called domU) can be equipped with virtual routers.

The fact, that Xen controls every privileged action of the domains results in two possible scenarios regarding packet forwarding [6]:

- each domU executes the forwarding itself
- dom0 bears all the forwarding activities

In both of the scenarios, the packets have to be allocated to the belonging domU out of the network stream and reverse. Dom0 sits behind the physical Network Interfaces (*Network Interface Cards*) and forwards the packet to the belonging virtual Interface of the domUs. So there is a lot of packet traffic between domU (what appears as the real network interface to the outer devices) and dom0, which has to be coordinated.
This can be done by a *bridging* (default) or a *routing mechanism*. In bridging a software bridge within dom0 executes this task and in routing IP-Adresses are assigned to the physical interfaces, as well as to the virtual interfaces within dom0.
In [6] several tests with different scenarios and combinations are carried out in competition with a single native linux. The authors draw the conclusion, that routing the packets through dom0 is the better solution (in contrast to bridging), especially with increasing number of domU's. Furthermore they found out, that handing over the forwarding function to the domU's results in a great performance loss. So when the forwarding task is operated by dom0 with a routing mechanism, then and only then Xen is able to forward packets as quick as a native linux. Surprisingly, the number of running virtual routers does not have any impact on the performance in this scenario. Finally the authors concluded, that when

doi: 10.2313/NET-2011-07-2_12

these conditions are taken into account, Xen is suited for the application as a virtual router platform.

In [2] another test scenario is evaluated by the use of a Click Router in combination with Xen. Click is a modular Open-Source Software-Router, implemented as Linux Kernel Add-On [8] with multi-threading support. In several scenarios it is analyzed, how fast packets are forwarded by the Click-Xen combination. Here it is experienced, that the smaller the forwarded packets are, the lower becomes the forwarding rate (measured in Gb/s). This is caused by the amount of memory accesses, that small packets are generating. This bottleneck can indirectly be handled by the CPU. CPU core switching during handling one packet and the triggered memory accesses can be avoided by allocating each virtual router to one particular core. Unfortunately this is the only way to influence the memory access bottleneck.

Another big challenge is sharing a *Network Interface Card* (NIC). Hardware-based allocation (every packet to the appropriate virtual router) is not fully supported neither by Click nor by Xen. But in contrast to the scenario above, here no software-based allocation is used. In the performed tests in [2] it is simply assumed, that the NICs are able to handle this demultiplexing. So the tests here are targeted a bit different. The results show, that core allocation becomes very difficult, when forwarding paths have different forwarding costs. Forwarding costs are for example composed of table lookups and the packet size. This issue is solved by a complex extended CPU Scheduler [2, page 5]. But as the authors appositely presume, sharing a single core might not be the issue in future, because development heads to increasing numbers of cores in CPUs. So it is concluded, that even today a virtual router can be realized using Xen and Click with the given problems to be solved in future. A similar project, just to be mentioned, is Trellis, where the same conclusion is drawn [9]. Finally, also in [3] the conclusion is drawn, that Xen is qualified for deploying "network-centric services".

4. OPENFLOW AND FLOWVISOR

The OpenFlow standard [10] is another possibility to enhance the possibilities of given networks. This open standard runs as an AddOn on Ethernet switches and routers and primarily separates the data path from the control path. The network is programmable and allows administrators to individually control and channel their data. Therefore only one single control unit is needed to administrate multiple routers and switches. This communication is carried out via a special OpenFlow Protocol, which has the benefit of beeing independent from hardware vendors. At the moment major device vendors are implementing OpenFlow in their hardware. OpenFlow is already be seen as a network virtualization technology by some members of the ONF, because it provides similar benefits. But this depends on the definition and point of view, of course. But to clarify: no multiple virtual router instances are running on the devices, like in Xen. To achieve real Network Virtualization FlowVisor is used [Section 4.2].

4.1 OpenFlow - Experimental Approach and Functionality

The most popular use case is the experimental test of new network protocols. This can be pefectly established in campus networks [10], because the researchers here can use their familiar environment and profit from the OpenFlow Roll-out at campus networks. The goal is to run experiments in the existing campus infrastructure without affecting the everyday work of others. Therefore a normal Computer is not sufficient, because of the low paket forwarding rate and the small number of ports. Consequently there is a need for an idealized OpenFlow Switch (Figure 4). This switch meets the demands of high-performance, low-cost, isolation and commercial vendors needs.

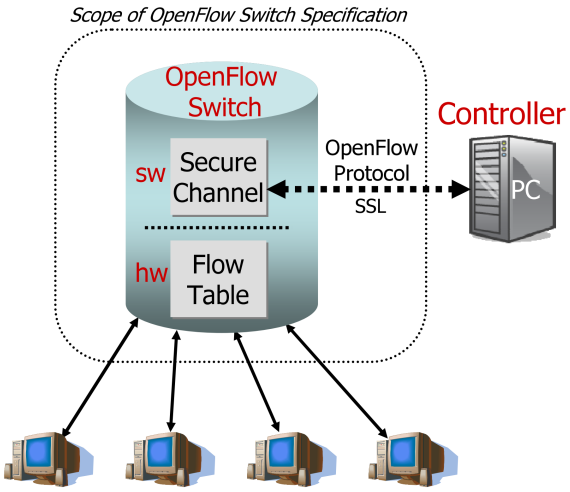

Figure 4: structure of an OpenFlow switch, showing the OpenFlow Software running above the hardware, modifying the FlowTable and communicating with the *Controller* [10]

The data path is still a task of the switch hardware, based on the flow table (provided by the OpenFlow controller). Here OpenFlow provides an interface to modify the flow tables of commercial router or switch products, exploiting the fact of many identical functions in the different products. As a result the experimental network traffic can be segregated from production traffic. All in all this enables similar benefits to the introduced solution with Xen (experiment with new routing protocols, addressing schemes and even IP alternatives [10]). The control instance saves a kind of forwarding rules as flow entries in the flowtable, each containing how to handle specific packets. A normal procedure in an OpenFlow-based hardware is to forward packets, when a flow entry already exists. The OpenFlow Software then knows how to forward the packet to which port. Another case is, that the arriving packet is unknown. Then OpenFlow sends the packet to the Controller via the encrypted OpenFlow Protocol. The Controller then decides, whether to create a new entry in the flow tables and to send it back to the switch, or to drop the packet.
So in an example configuration with many switches and routers, when a flow is defined at the control unit a new protocol comes into operation and automatically creates a new route for the packets by entering automatically all relevant FlowEntries in each switch. This is called a flow.

doi: 10.2313/NET-2011-07-2_12

In OpenFlow there are two major opportunities to achieve traffic isolation:

- forward the packets through the switch's normal processing pipeline
- assign VLAN IDs to the groups of packets to allocate them to different VLANs

Many sample applications are given in [10, page 4], just as creating VLAN similar environments, establishing VoIP connections, defining new addressing, naming and routing schemes and even abstracting from flow processing of the controller to programmable router based processing.

4.2 FlowVisor Network Virtualization Layer

One step ahead goes FlowVisor, "a special purpose OpenFlow controller that acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers" [11]. So in a FlowVisor application field, when a packet arrives at an OpenFlow Switch, it is routed by the FlowVisor to the belonging Controller and reverse forwards the packets back to the switches. This is performed in an isolated way by assuring, that no resources, like the FlowTables can affect each other. So FlowVisor supplements OpenFlow by adding real virtualization possibilities. FlowVisor provides virtualization of switches to build a divided, fully isolated and autonomous network above the physical structure. Every network is logically independent and runs in addition to a productive network on the same hardware. In [12] these virtual networks are called *slices*. FlowVisor regards a slice as any combination of switch ports and layer 2, 3 and 4 of the OSI-model [14]. Of course, FlowVisor follows the virtual network approach and supports its advantages, like resource allocation or Checkpoint Saving [Section 1].

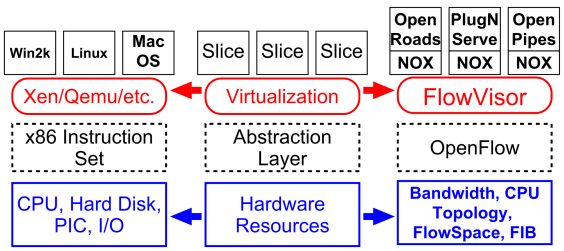

Figure 5: FlowVisor layer comparison to Xen and abstract virtualization - Flow Visor is located betweeen the OpenFlow Switch Software and the Controllers above. Open Roads, PlugN and Open Pipes are examples of virtual network controllers [12]

The layer location of FlowVisor in Figure 5 shows, that it is comparable to other virtualization technologies and of course to Xen, we treated before. Flow Visor is located between the OpenFlow Switch(forwarding path) Software and the Controllers (control path) above. Open Roads, PlugN and Open Pipes are examples of virtual network controllers [12]. To the controllers FlowVisor appears as a set of OpenFlow Switches and to the OpenFlow Switches it appears as a set of Controllers. FlowVisor itself hosts multiple OpenFlow guest controllers, as shown in Figure 6, one for each virtual network. The design focuses here on strong isolation. This involves the controllers of the different slices, as well as the belonging datapath traffic. One slice should not be able to influence another one. This Isolation is achieved by following which flow entry to which controller belongs and assigning a minimum data rate to a slice [10]. FlowVisor should be transparent (with the meaning of imperceptible) for the controller and the OpenFlow Switch. This is for example important when an error or bug occurs in a test environment. It simplifies the finding and fixing process. Furthermore FlowVisor should support resource allocation. This is implemented by a special module, called the *Resource Allocation Policy* [12].

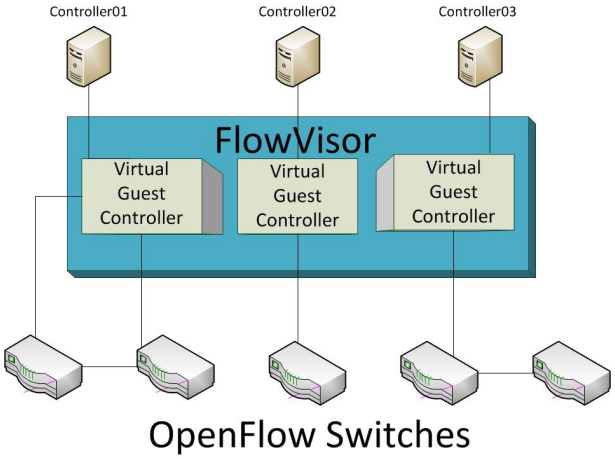

Figure 6: FlowVisor internal structure - FlowVisor hosts multiple virtual guest controllers, each responsible for one virtual network

The three major design goals of FlowVisor are summed up in [12] as follows:

1. Transparency
2. Isolation
3. Extensible Slice Definition

4.2.1 *Isolation and Challenges*

Since isolation is one of the critical issues in Network Virtualization, this Section treats the isolation capabilities of FlowVisor. *Bandwidth Isolation* is only provided with the instruments of VLANs. Each packet has a VLAN Priority Code Point field, where an entry assigns a packet to a certain priority level. Combined with the traffic class in the Resource Allocation Policy this enables a kind of bandwith allocation. But the big disadvantage is, that the exact specifications of a traffic class must be manually implemented in each switch. This shows, that in this field future research has to be done to make this issue more practicable. *Topology Isolation* means, that FlowVisor transmits only information of the belonging switches to each of its virtual guest controllers inside. So each virtual network, or each guest controller gets only information about his own network. Due to the fact most switch hardware has low-performance CPUs, the risk of a breakdown of the OpenFlow Software is very high on an overload. FlowVisor does not support CPU Isolation at the

doi: 10.2313/NET-2011-07-2_12

moment. Only a few workarounds exist, which explain how to deal with the limited CPU. Furthermore the *FlowSpace Isolation* assures, that one virtual guest controller can only affect the own virtual network with its created rule. Even the connection to the OpenFlow controller is virtualized and controlled by transaction IDs. This ensures, that no guest controller can block or even catch a transaction of another guest controller.

4.2.2 Performance

As mentioned in Section 1, adding a additional virtual layer between two instances results in performance deficits. Of course this is for FlowVisor as well true as for any other virtualization technology. The goal rather is to reduce this performance overhead to a negligible amount. FlowVisor realizes this through only acting in situations where it is really necessary. All data and control paths work at full line rate, without beeing slowed down by FlowVisor. This also applies for any route selection, carried out by a controller. The only situation when FlowVisor intervenes, is when a new flow message is send by the switch (a new unknown packet arrived) and port status messages (controller demands switch to send byte and packet counters for a specific port). The tests in [12] show, that the average overhead of a port status request is about 0.483ms and for flow messages 16.16ms. So we see the latency for port status request is quite acceptable, where in contrast the 16.16ms latency is probably a bit too high for time sensitive applications.

5. CONCLUSION

Our goal was to give an overview over Network Virtualization and to highlight the ability of Xen and OpenFlow/ FlowVisor to realize this approach. So all-in-all the topic of Network Virtualization is still an experimental field. But recent developments show, that the industry is interested in deploying this technique in praxis [Section 1]. What solution has the best chances to become successfully deployed in large scale networks, depends on many factors and can not be clearly identified. Here future research is needed. So Xen provides a relative stable technology yet, while the quite young OpenFlow Project now receives great support by the ONF. But a disadvantage of FlowVisor is, that at this time the software itself is not stable enough to be transferred into production. Given the quite big performance overhead of the flow messages, we can summarize, that a lot of work has to be done in future. With the given constraints, Xen can even nowadays provide a functional platform for virtual routers and networks. Due to the increasing interest in programmable, virtual networks, this is going to be a subject of interest in the future. Especially science will benefit from this demand in research.

6. REFERENCES

[1] Jorge Carapinha and Javier Jimnez: *Network Virtualization – a View from the Bottom*, VISA '09 Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, pages 1-3, ACM New York, NY, USA ©2009

[2] Norbert Egi, Laurent Mathy, Mickael Hoerdt, Adam Greenhalgh, Mark Handley and Felipe Huici: *Fairness Issues in Software Virtual Routers*, PRESTO '08 Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow, ACM New York, NY, USA ©2008

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt and Andrew Warfield: *Xen and the Art of Virtualization*, SOSP '03 Proceedings of the nineteenth ACM symposium on Operating Systems principles, ACM New York, NY, USA ©2003

[4] Diwaker Gupta, Ludmila Cherkasova, Rob Gardner and Amin Vahdat: *Enforcing Performance Isolation Across Virtual Machines in Xen*, PROCEEDING - Middleware '06 Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware , Springer-Verlag New York, Inc. New York, NY, USA ©2006

[5] L. Seawright and R. MacKinnon: *VM/370 - A Study of Multiplicity and Usefulness*, IBM Systems Journal, pages 4-17, 1979

[6] Norbert Egi, Adam Greenhalgh, Mark Handley, Micka"el Hoerdt, Laurent Mathy and Tim Schooley: *Evaluating Xen for Router Virtualization*, Proceedings of 16th International Conference on Computer Communications and Networks, Computer Communications and Networks, 2007. ICCCN 2007, Honolulu, HI ©2007

[7] Gerald J. Popek and Robert P. Goldberg: *Formal requirements for virtualizable third generation architectures*, Commun. ACM, 17(7):412–421, 1974.

[8] Daniel Schwencke: *Click – ein modularer Router*, Seminary paper, TU Braunschweig, July 2006

[9] Sapan Bhatia, Murtaza Motiwala, Wolfgang M"uhlbauer, Vytautas Valancius, Andy Bavier, Nick Feamster, Larry Peterson and Jennifer Rexford: *Hosting Virtual Networks on Commodity Hardware*, WORKSHOP ON REAL AND OVERLAY DISTRIBUTED SYSTEMS (WORLDS), 2008

[10] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker and Jonathan Turner: *OpenFlow: Enabling Innovation in Campus Networks*, Newsletter ACM SIGCOMM Computer Communication Review archive, Volume 38 Issue 2, April 2008, ACM New York, NY, USA

[11] *FlowVisor: Home*, `https://openflow.stanford.edu/display/flowvisor/Home` accessed on May 24th, 2011

[12] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown and Guru Parulkar: *FlowVisor: A Network Virtualization Layer*, OPENFLOW-TR-2009-1

[13] Christian Kern: *Paravirtualisierung, Vanderpool*, Haupseminar Virtualisierungstechnologien, Technische Universität München, Institut für Informatik, Lehrstuhl für Rechnertechnik und Rechnerorganisation, Prof. Dr. Arndt Bode, 22.07.2005

[14] Hubert Zimmermann: *OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection*, IEEE TRANSACTIONS .ON COMMUNICATIONS, VOL. COM-28, NO. 4, APRIL 1980

doi: 10.2313/NET-2011-07-2_12