Implementing Network Virtualization for a Future Internet

Panagiotis Papadimitriou[§] Olaf Maennel[‡] Adam Greenhalgh[*] Anja Feldmann[‡] Laurent Mathy[§]

[§]Computing Dept., Lancaster University, UK
{*p.papadimitriou, l.mathy*}*@lancaster.ac.uk*
[‡]Deutsche Telekom Laboratories, Germany
{*olaf, anja*}*@net.t-labs.tu-berlin.de*
[*]Dept. of Computer Science, University College London, UK
a.greenhalgh@cs.ucl.ac.uk

***Abstract*— The Internet has become an essential communication medium upon which billions of people rely every day. However, necessary evolution of the Internet has been severely limited by reliability constrains and social-economic factors. Experts fear that current growth rates will threaten the future of the Internet as a whole, despite the fact that new core technologies already exist.**

Network virtualization presents a promising approach to overcome ossification and facilitate service deployment for a future Internet. Exploring this approach, we present a prototype implementation which realizes a subset of the 4WARD virtual network (VNet) architecture, allowing multiple VNets to coexist on top of a shared physical infrastructure. We discuss the functionality of our prototype and demonstrate experimental results to assess its performance.

I. Introduction

The Internet realizes William Gibson's vision of Cyberspace: "A consensual hallucination experienced daily by billions of legitimate operators, in every nation". Indeed, this so-called 'network of networks' has changed the way that humans interact and provides the crucial foundation for a multitude of businesses.

To residential customers or service providers the Internet appears as widely successful and well-performing; nevertheless, the underlying infrastructure suffers from ossification [15], [4]. Service Level Agreements (SLAs), for example, demand high reliability constrains, such as 99.99999% network availability. Short outages, already in the order of minutes per year, or a router misconfiguration can easily cause serious implications for the Internet Service Provider (ISP). This impacts the deployment of new services, as the risks of breaking existing services are high. Therefore, according to the saying "never change a working system", ISPs often have no incentive to change their network or to introduce new technologies.

Indeed, over the last 20 years most new disruptive technologies came from the edge, while the core has remained almost unchanged. However, this resistance to change leads to some serious problems. Soon ISPs might not be able to add any more new customers, because IPv6 deployment has been neglected and we are now facing the depletion of the IPv4 address space. Also routing table growth, inadequate support for inter-provider multicast, Quality of Service (QoS), device mobility, manageability and/or security will eventually enforce providers to revisit their core architecture.

To circumvent the difficulty of changing successful networks, the concept of overlays has proven to be very useful [16]. One of the key insights when looking at an overlay network is that each overlay can be considered to be a virtual network. Fundamentally, virtualization is an *abstraction concept* in that it hides details, and therefore allows you to cope with heterogeneity and complexity. As such it offers a *level of indirection* as well as *resource sharing*. The former results in more flexibility while the latter enables partitioning as well as the re-usage of physical resources, resulting in higher efficiency. However, to ensure success *resource separation* with a sufficient level of *isolation* is required.

In this paper, we explore the space of network virtualization by utilizing existing technologies. Our primary goal is to demonstrate that already today we have all the necessary ingredients needed to create a paradigm shift towards full network virtualization. We present the facets of how to create VNets, and discuss the lessons learned from implementing a prototype system. This serves as an example of how existing technologies can be leveraged to help take the first steps towards an evolvable Internet.

The remainder of the paper is organized as follows. Section II outlines the enabling technologies required to build a virtual network. In Section III, we present the architecture behind our prototype implementation. In Section IV, we discuss the functionality and some implementation details for our prototype. Section V provides experimental results with the prototype. Finally, in Section VI we highlight our conclusions and refer to directions for future work.

II. Enabling Technologies

In this section, we provide an overview of the various link and node virtualization technologies that exist today and could be used to form a virtualized network.

Virtual Local Area Networks (VLANs) are the most common example of *link virtualization*, allowing the creation of isolated networks where all participating nodes are in a single broadcast domain. Most Ethernet switches support tagged and untagged VLANS based on the IEEE 802.1Q standard, whilst some support the new stacked VLAN tags standard based on IEEE 802.1ad. Besides VLANs, tunneling technologies, such as encapsulation of Ethernet frames to IP datagrams (EtherIP) [9], IP-in-IP, *Generic Routing Encapsulation* (GRE) [7] and *Multi-protocol Label Switching* (MPLS) [17], are widely used to create virtual private networks (VPNs) and eventually build the foundation for virtual networks in Wide Area Networks (WAN).

Node virtualization aims to provide efficient sharing and isolation of computing resources, such as CPU and memory, so that multiple operating systems can run concurrently in a single physical machine. Existing techniques for virtualizing physical nodes include *full virtualization*, *paravirtualization* and *container-based* virtualization. Full (or hardware) virtualization, such as KVM [10], provides a fully emulated machine in which a guest operating system can run. Full virtualization offers the highest level of isolation; however, it may incur performance and resource penalty. Paravirtualization mechanisms, such as *Xen* [2] and *Denali* [19], provide a *Virtual Machine Monitor* which runs on top of the hardware and is able to host multiple virtual machines (VM). Each VM appears as a separate computer with its own operating system and software. Paravirtualization, therefore, offers increased flexibility when different *guest* operating systems are required to run in a physical node. However, this capability results in increased overhead compared to container-based virtualization. The latter approach creates multiple partitions of operating system resources, called *containers*, relying on advanced scheduling for resource isolation. However, the level of achievable isolation is usually lower in comparison with paravirtualization mechanisms. In addition, each container is bound to run the same version of operating system as the host machine. The obvious advantage of container-based virtualization is performance due to their reduced overhead. Typical examples of this virtualization technology are OpenVZ [13] and Linux-VServer [12].

III. Network Virtualization Architecture

Visions of a future Internet, and in particular network virtualization architectures, have been discussed for more than 10 years (e.g., [18], [1], [3], [6], [20]). In this section, we briefly summarize the architectural concepts behind our prototype implementation.

A. Roles and Actors

Today's Internet is maintained and operated by a set of Internet ISPs and companies. Each ISP owns its part of the physical infrastructure and operates an IP network on top of that infrastructure. The Internet is created by inter-connecting all of these networks. However, there is also a "vertical" split:

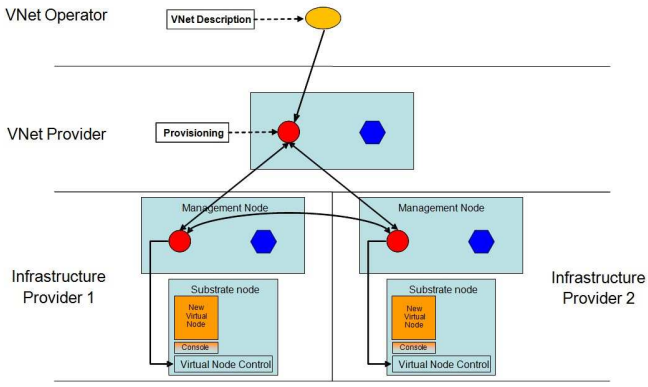


Fig. 1. VNet Architecture Overview

one part of the ISP is often responsible for owning and maintaining the physical resources (e.g., routers, switches), while a different part of the same ISP is responsible for providing IP-layer connectivity (e.g., managing configuration, and assuring routing is functioning). Network virtualization adds a layer of indirection between physical resources and network operation and management. Hence, virtualization provides the technical prerequisite for a technical separation of these business roles.

The VNet architecture (Fig. 1) enables multiple VNets to coexist on top of a shared physical infrastructures spanning multiple infrastructure providers. Such VNets can be tailored to have certain properties and guarantees not currently available in today's Internet, such that existing or emerging applications and services can be more effectively deployed. This architecture comprises the following actors (Fig. 1):

- The Physical Infrastructure Provider (PIP), which owns and operates the physical network, the physical routers and the physical interconnects with other PIPs. The VNet architecture assumes the existence of multiple PIPs which can lease slices of their physical resources in order to enable the creation of multiple isolated virtual networks.

- The Virtual Network Provider (VNP), which combines the slices of virtualized infrastructure from PIPs together into a functional VNet. This provides the missing layer of indirection between the PIP and the VNO.

- The Virtual Network Operator (VNO), which operates and manages the instantiated VNets. For the VNO the virtualized networks appear as if they were physical networks. The VNO does not necessarily have any knowledge of the substrate configuration or the PIP. A VNO can only control resource allocations indirectly by specification of requirements that must be met by the VNP.

These roles reflect the indirection created by the virtualization mechanism. Certainly, the roles of the PIP and the VNP may be performed by the same business entity or the VNP

and the VNO may coincide. In the remainder of this section, we discuss VNet instantiation and end-user attachment.

B. Virtual Network Instantiation

VNet instantiation involves several interactions between the VNO and the VNP, as well as the VNP and the participating PIPs. All VNet requests are communicated using a (yet to be standardized) resource description model which includes separate descriptors for nodes and links, allowing ultimately a coherent VNet specification.

Each PIP exposes interfaces to allow the VNP to request virtual resources and networks. VNet instantiation requires at least one (pre-determined) *management node* within each PIP. Such a node is mainly responsible for handling VNet requests on behalf of the PIP. Essentially, the instantiation of a VNet takes place in the following consecutive steps:

1) Exchange of resource information: VNet requirements are formulated and handed from the VNO to the VNP. Such requirements describe the number of virtual resource instances, as well as their respective properties. Since VNet requests are not known in advance, they are processed as they arrive both by the VNP and the PIPs. The outcome of each request is communicated to the VNP and subsequently to the VNO.

2) Resource discovery: The VNP maps the requested virtual resources and their properties onto PIPs, negotiates with the PIPs for resources and interconnections and hands respective partial topology descriptions to them. Each PIP is responsible for the mapping of the given partial topology onto its substrate and the setup of connections to PIPs hosting neighboring parts. Alternatively, a PIP may be willing to advertise its available physical resources, delegating the mapping of the VNet to the VNP. In this case, the PIP eliminates the mapping overhead with the potential risk of revealing resource information.

3) Resource virtualization: Following resource discovery and VNet mapping, each PIP management node is in charge of the virtualization process by signaling individual requests to the assigned substrate nodes. Each node handles such a request within its management domain, which subsequently creates and configures virtual machines.

4) Topology construction: Virtual nodes are typically interconnected by setting up tunnels on top of the substrate in order to form the requested virtual network. Each virtual machine uses its virtual interface to transmit packets, which are encapsulated and injected to the tunnel. On the receiving host, the incoming packets are demultiplexed and delivered to the appropriate virtual machine.

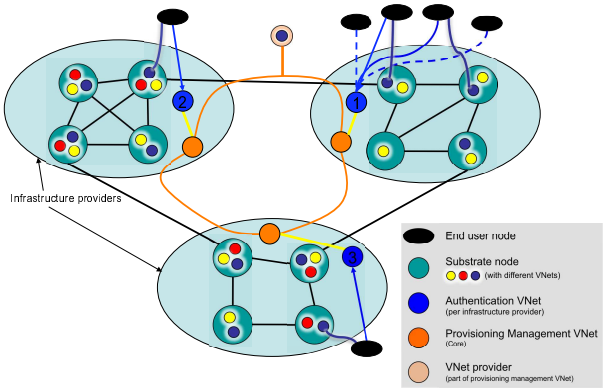


Fig. 2. VNet End-user Attachment

5) Console access: In order to allow the operation and management of the virtualized infrastructure, the VNP establishes console access to the instantiated virtual nodes.

This sequence of steps provides a fully virtualized network which is ready to be operated and managed by the VNO.

C. End-user attachment

Upon VNet instantiation, the end-user needs to establish connection to the VNet. End-user attachment can be achieved by the following two options:

- The VNO/VNP issues a request to extend the VNet all the way to the end-user.
- The end-user requests that a tunnel over existing substrate is constructed towards a “VNet access point”.

According to the first option, the VNet is provisioned all the way to the end-user in the same way as the whole virtual network was set up. This means that VNet provisioning is triggered by the VNO, communicated via the management system to VNP, and subsequently requested as dedicated resources from the PIP. The main advantage of this approach is that all QoS guarantees, and security requirements are met. Disadvantages include scalability and the fact that nodes in the access network of the PIP need to support virtualization.

Alternatively, the user can initiate the connection setup, as shown in Fig. 2. In contrast to the previous approach, this option offers significant scalability advantages, especially with a large number of end-users. In this case, the end-user needs to establish connectivity with the substrate, which subsequently requires authentication with the PIP while the user has to know or retrieve the “VNet access point”. Provided that the PIP is able to provide substrate connectivity up to that point, end-user attachment is eventually achieved via a “tunnel” or “VPN”. Using a tunnel, QoS guarantees are much harder to fulfill; however, the legacy equipment on the last mile renders this approach more scalable.

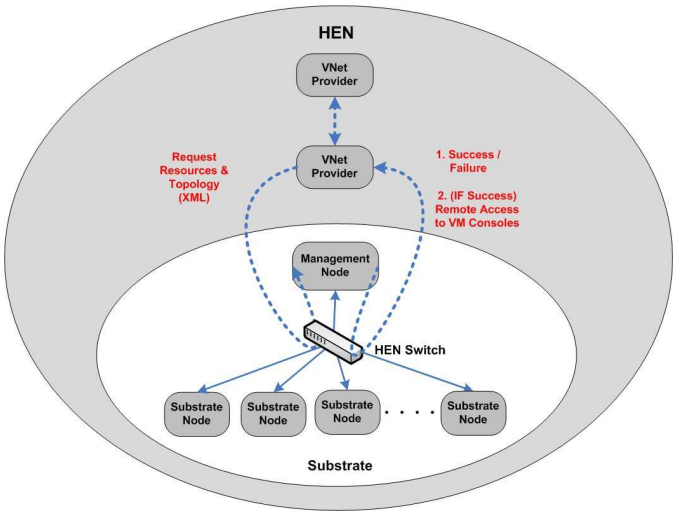


Fig. 3. Prototype Overview

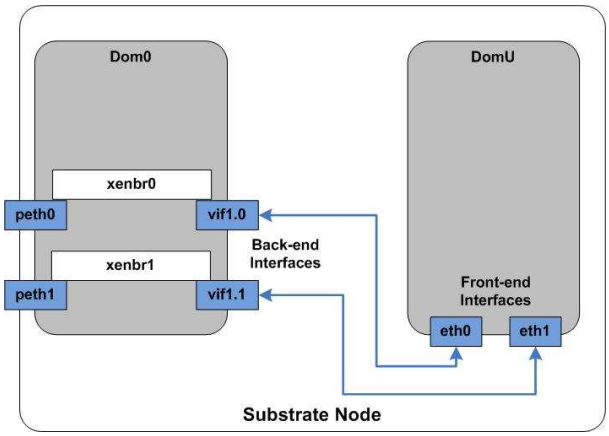


Fig. 4. Substrate Node with Bridged Configuration

IV. Prototype Implementation

In this section, we discuss the implementation and functionality of a prototype for the preceding architecture.

A. Infrastructure and Software

The prototype is implemented on *Heterogeneous Experimental Network* [8], which includes more than 110 computers connected together by a single non-blocking, constant-latency Gigabit Ethernet switch. We mainly used Dell PowerEdge 2950 systems with two Intel quad-core CPUs, 8GB of DDR2 667MHz memory and 8 or 12 Gigabit ports.

The prototype takes advantage of node and link virtualization technologies to allow the instantiation of VNets on top of a shared substrate. We used Xen 3.2.1, Linux 2.6.19.2 and the Click modular router package [11] (version 1.6 but with patches eliminating SMP-based locking issues) with a polling driver for packet forwarding. We relied on Xen's paravirtualization for hosting virtual machines, since it provides adequate levels of isolation and high performance [5].

B. Functionality Overview

We implemented all the basic functions of the infrastructure, the VNP and VNO, as described in Section III, so that the prototype can realize the instantiation of VNets. Furthermore, the prototype offers on-demand operational and management capabilities (e.g., virtual machine migration), which are typically invoked by the VNO.

A fixed number of HEN physical nodes compose the substrate (PIP) which offers resources to the VNP for on-demand creation of virtual networks. Separate nodes act as the network operation centre (NOC) of the VNP and VNO. The NOC of the VNP establishes direct connection with a dedicated management node belonging to the PIP. This node is mainly responsible for realizing all the VNet requests to the PIP. Fig. 3 illustrates an overview of this prototype.

The communication between the NOC of the VNP and the substrate management node involves the exchange of resource information. We used an XML schema for the description of virtual resources with separate specifications for nodes and links. Our resource description is able to capture a variety of VNet requests, such as location, processing power, memory, link bandwidth or number of network interfaces, allowing the instantiation of a network according to the requested specification.

Since VNet requests are not known in advance by the substrate, the prototype allows the dynamic processing and execution of these requests (subject to availability of physical resources) as they arrive. The outcome of each request is subsequently communicated to the VNP. Our prototype periodically monitors the state of the substrate resources enabling the management node in the substrate to check whether the available physical resources are sufficient for the execution of a VNet request. This function allows for admission control when the substrate resources are limited, i.e., rejecting (or postponing) VNet requests when they violate the resource guarantees for existing virtual nodes or networks.

Our implementation supports resource discovery either at PIP (i.e., resources are not advertised) or at the VNO (i.e., when VNO is aware of physical resources). Besides the availability of physical resources, the PIP maintains a list of individual attributes for its nodes, such as the number of available network interfaces. This information is useful when assigning substrate resources to virtual network components.

Upon resource discovery, the PIP management node signals individual requests to the substrate nodes, which are handled by their management domain (Dom0). The prototype supports two options for node virtualization: (i) the virtual machines are created and booted on-demand as guest domains (DomUs), or (ii) the PIP has allocated CPU and memory to virtual machines in advance, avoiding their creation and booting upon receiving a VNet request. With the second option, a VNet can be instantiated very fast; however, physical resources are wasted when virtual machines remain unused. The prototype also allows certain configuration options for the instantiated virtual machines. These options can be part of a VNet request and mainly include the number of physical interfaces that will be attached to each virtual node and whether a bridge or Click

will be used for this purpose.

For the inter-connection of the virtual nodes, we currently use tunnels with IPv4-in-IPv4 encapsulation. However, in principle our implementation is not limited to IP forwarding, neither to tunnels. Other link-virtualization mechanisms, such as MPLS, could be used with our prototype. Further details on how we set up link virtualization are given in the following subsection.

After VNet instantiation, the prototype allows for on-demand configuration and management operations, such as termination of existing virtual machines, the attachment of additional physical interfaces, topology modifications and migration of virtual machines from one physical node to another.

C. Implementation Details

We hereby refer to some details behind our implementation. Fig. 4 provides a more detailed view of a substrate node, illustrating the interaction between the management (i.e., Dom0) and the guest domains (i.e., DomUs). Dom0 acts as the driver domain for the physical hardware, including the network interface cards (NIC). Xen splits a NIC driver into a front-end and back-end driver. The front-end resides in the kernel space of the guest operating system (i.e. in DomUs) while the back-end is part of the Dom0 kernel. Each portion of the network driver creates a corresponding virtual device interface, as shown in Fig. 4. The device interfaces in Dom0 are represented as *vifX.Y*, where X and Y correspond to the unique domain and interface IDs, respectively. Each DomU includes a front-end interface which essentially appears as a real network interface (i.e., ethY) in the particular guest domain. Xen creates I/O channels between a Dom0 and each instantiated DomU connecting their corresponding back-end and front-end interfaces. In this sense, any packet sent through the back-end interface appears as being received by the front-end device in the DomU and vice versa. Upon creating an I/O channel, Xen bridges the respective back-end interface onto a physical NIC.

The standard Xen configuration results in bridging all the existing back-end interfaces (that correspond to separate DomUs) onto a single physical NIC. Such configuration may be undesirable due to the complexity incurred by sharing a single NIC among multiple DomUs. Furthermore, a misconfiguration in packet forwarding may cause packets traversing the shared bridge to be intercepted by any DomU that uses the same bridge. The prototype provides the flexibility of attaching separate NICs per DomU. In this case, the management domain configures an additional pair of front-end and back-end interfaces and subsequently creates a Xen I/O channel to establish their communication. This automated process is concluded by bridging the back-end interface onto an available NIC. Therefore, packets generated by a guest domain easily find their way to the physical network while incoming packets can be also received by the DomU. Alternatively, we provide the option of packet forwarding between a pair of front-end and back-end interface using Click, in the case where bridging is not desired.

Upon receiving a request for terminating a guest domain which communicates with the preceding configuration, Dom0 destroys all corresponding device interfaces along with the bridge (or alternatively removes Click instances) and the I/O channel, relinquishing the NIC for the instantiation of a new guest domain. On the arrival of a migration request and in the presence of such configuration, the virtual machine is moved from one physical node to another as follows:

1) Xen is instructed to migrate the virtual machine to the new host.
2) The virtual devices, I/O channel and the required bridge are re-created in the new host.
3) The configuration in the previous host is not needed and therefore it is removed.

The prototype should achieve instant and transparent migration of virtual nodes, upon a request handed by the VNO or a load-balancing mechanism within the substrate. To prevent unnecessary delays when seeking the location of a virtual machine in a large number of substrate nodes we extended Xen with the capability of discovering the physical node that hosts a particular virtual machine.

The substrate topology is constructed by configuring VLANs in the HEN switch. This process is automated via a *switch-daemon* which receives VLAN requests and configures the switch accordingly. In reality, this step is not required, since each PIP provides a physical network topology on top of which a VNet (or part of it) can be directly mapped.

Virtual links are set up by encapsulating and demultiplexing packets, as shown in Fig. 5. More precisely, each virtual node uses its virtual interface to transmit packets, which are captured by Click for encapsulation, before being injected to the tunnel. On the receiving host, Click demultiplexes the incoming packets delivering them to the appropriate virtual machine. Substrate nodes that forward packets consolidate all Click forwarding paths in a single domain (Dom0) avoiding costly context switches; hence, the achievable packet forwarding rates are very high [5]. In all cases, Click runs in kernel space.

V. Evaluation

In this section, we demonstrate experimental results that show the efficiency of the prototype implementation and give an early glimpse on the feasibility of the VNet architecture. In particular, we evaluate (i) VNet instantiation and (ii) packet forwarding performance. At the same time, we demonstrate the performance of low-cost commodity hardware as network devices. In the near future, we believe that more ISPs will replace expensive routers in their access or aggregation network with PC hardware which already exhibits good forwarding performance and scaling properties. For our experimentation, we use Dells PowerEdge 2950 with specifications as described in Section IV-A. We use separate nodes for the PIP (including the management node), the VNP and the VNO.

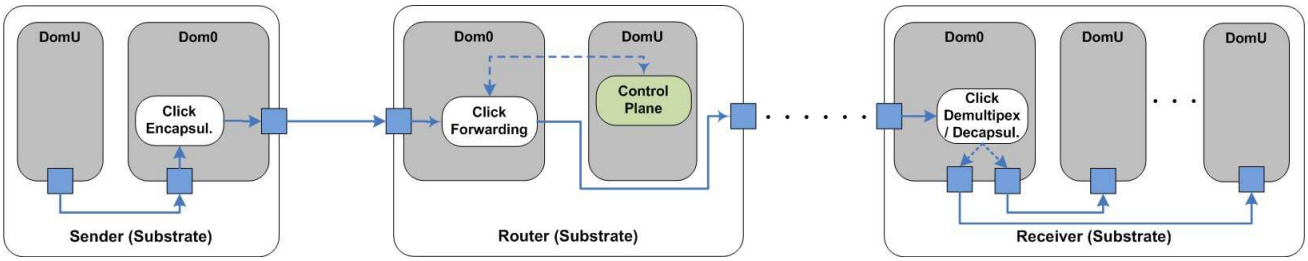


Fig. 5. Virtual Links on the Prototype

A. VNet Instantiation

The instantiation of VNets composes a critical procedure for this architecture. With resource discovery being an important element in virtualized environments, we consider the following scenarios:

1) Discovery at PIP: The VNP is not aware of the substrate resources and the PIP maps the requested VNet to the substrate.

2) Discovery at VNP: The PIP advertises its physical resources and subsequently, the VNP determines how the requested VNet will be mapped to the substrate.

Initially, we measure the time required to instantiate the VNet of Fig. 6, including resource discovery, the creation and booting of virtual machines, setting up the tunnels and console access to the virtual nodes. Table I provides the corresponding measurements for both resource discovery scenarios. Our results reveal that, in both cases, instantiation times are reasonable, with most time being spent within the PIP.

TABLE I

VNET INSTANTIATION TIME (SEC)

	min	avg	max	stddev
Resource discovery at PIP	103.38	109.47	119.27	4.43
Resource discovery at VNP	104.08	110.37	120.79	4.27

With resource discovery within VNP, instantiation is slightly slower, since further interactions between the VNP and the PIP are required. More precisely, the PIP management node has to communicate its resources to the VNP, which in turn initiates the mapping and subsequently instructs the PIP how to assign physical resources to virtual network components.

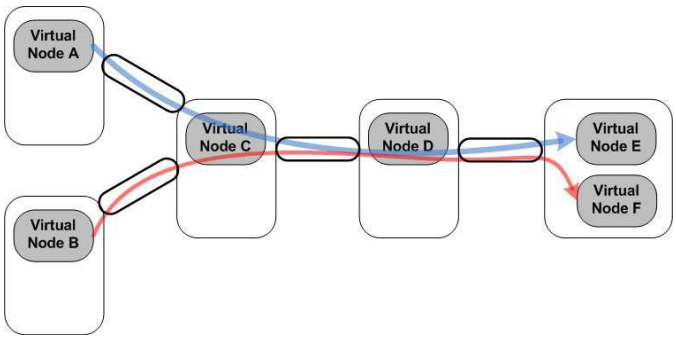


Fig. 6. Experimental Topology.

So far, VNet instantiation is dominated by the time required to create and boot virtual machines. As mentioned in Section IV-B, our prototype provides the flexibility to allocate CPU and memory resources to virtual machines in advance and hence assign them to a VNet upon its request. Table II contrasts VNet instantiation times with pre-allocated virtual machines to the respective instantiation times with on-demand creation (from Table I). It is clear that virtual machine pre-allocation results in remarkably faster VNet instantiation unfolding the potential of the VNet architecture.

TABLE II

VNET INSTANTIATION TIME (SEC)

	min	avg	max	stddev
On-demand VM creation	103.38	109.47	119.27	4.43
VM Pre-allocation	15.72	16.75	17.59	0.41

We further use OProfile [14] to monitor CPU utilization for the substrate nodes and VNP during VNet instantiation. Table III shows the corresponding measurements with on-demand virtual machine creation and resource discovery at the PIP. According to these results, our prototype implementation does not impose an unreasonable overhead either to the substrate or the VNP. We have also obtained similar measurements with the rest of instantiation scenarios discussed above. Although these results are specific to our implementation, they show that VNet instantiation is technically feasible.

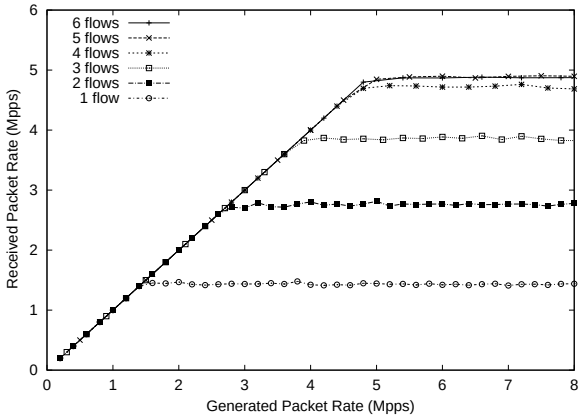
TABLE III

%CPU DURING VNET INSTANTIATION

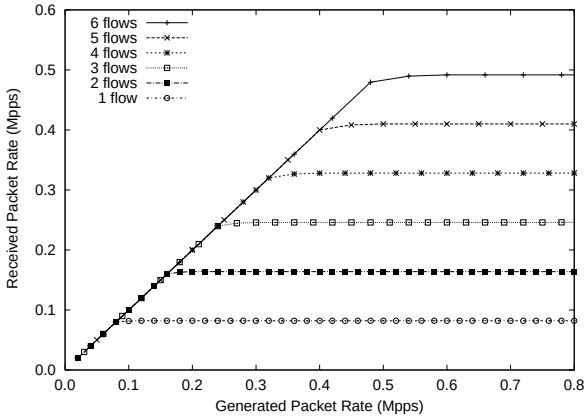
	min	avg	max	stddev
VNP	20.38	23.45	25.49	1.52
Substrate Node	16.33	19.15	22.61	2.05

B. Packet Forwarding Performance

We hereby assess the packet forwarding performance achieved by our virtualized data planes. Our primary goal is to show that virtual data planes do not impose considerable restrictions in terms of packet forwarding when the right design is followed. Recall that all forwarding paths are consolidated in Dom0, which therefore acts a common forwarding domain. Fig. 7 demonstrates aggregated forwarding rates with



(a) 64-byte packets



(b) 1500-byte packets

Fig. 7. Aggregated Packet Forwarding Rates.

1 to 6 unidirectional flows and 64- and 1500-byte packets. In each case, packet forwarding involves an IP lookup. The experiments were conducted on a Dell PowerEdge 2950 with 12 physical interfaces and 8 CPU cores, with each flow assigned to separate interfaces and cores.

According to Fig. 7(a), the aggregated forwarding rate for 6 flows and 64-byte packets nearly reaches 5 Mpps, which is remarkable for non-specialized hardware. The forwarding performance scales well up to 4 forwarding paths; from this point, memory access latency limits forwarding rates. A thorough exploration of this performance bottleneck can be found in [5].

Forwarding large packets allows us to evaluate both forwarding performance (Fig. 7(b)) and link utilization (Fig. 8). In this case, forwarding rates scale perfectly with the number of forwarding paths, while the throughput achieved always reaches the line rate, resulting in full utilization of the available bandwidth.

VI. Conclusions and Future Work

In this paper, we initially presented an overview of the 4WARD VNet architecture uncovering which technological ingredients are necessary for its implementation and how they have to be combined to provision and operate VNets. Our main contribution is the implementation of the architecture prototype which realizes the instantiation and on-demand configuration of multiple VNets on top of the shared physical infrastructures. We further presented experimental results with our prototype showing that the instantiation of VNets is technically feasible. Our measurement results also demonstrate that virtualized data planes on commodity hardware are capable of fulfilling the forwarding requirements of aggregation routers in the access part of an ISP's network and thus providing a serious and inexpensive alternative to commercial routers.

After studying the feasibility of VNet instantiation, our implementation can be used to provide insights into the architectural design decisions and help understand the advantages and disadvantages of them. For example, did we actually create the right tussle boundaries? What are the trade-offs between technological constraints and business goals? We further plan to investigate the timescales on which VNets can be requested or provisioned. Future work also includes the enhancement of our resource description language.

VII. Acknowledgments

We are thankful to 4WARD partners and Mickaël Hoerdt for useful discussions and comments on this work.

Part of this work was performed within the 4WARD project, which is funded by the European Union in the 7th Framework Programme (FP7).

References

[1] 4WARD Project, http://www.4ward-project.eu.
[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R.Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization", in *Proc. 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, USA, October 2003.
[3] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI Veritas: Realistic and Controlled Network Experimentation", in *Proc. ACM SIGCOMM '06*, Pisa, Italy, September 2006.
[4] D. Clark, J. Wroclawski, K.R. Sollins and R. Braden, "Tussle in Cyberspace: Defining Tomorrow's Internet", in *Proc. ACM SIGCOMM '02*, Pittsburgh, USA, August 2002.
[5] E. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Towards High Performance Virtual Routers on Commodity Hardware", in *Proc. ACM CoNEXT 2008*, Madrid, Spain, December 2008.

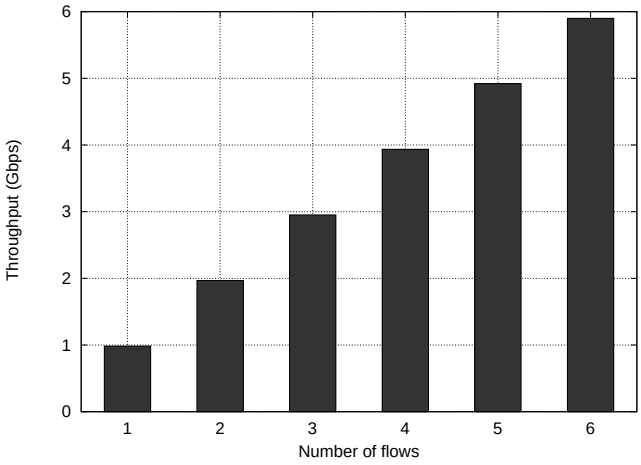


Fig. 8. Throughput with 1500-byte packets.

[6] FIND: Future Internet Design, http://www.nets-find.net/.
[7] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, “Generic Routing Encapsulation (GRE)”, RFC 2784, IETF, March 2000.
[8] Heterogeneous Experimental Network, http://hen.cs.ucl.ac.uk.
[9] R. Housley and S. Hollenbeck, “EtherIP: Tunneling Ethernet Frames in IP Datagrams”, RFC 3378, IETF, September 2002.
[10] A Kivity, Y Kamay, D Laor, U Lublin, and A Liguori, “KVM: The Linux Virtual Machine Monitor”, in *Proc. Linux Sysmposium*, Ottawa, Canada, 2007.
[11] E. Kohler, R. Morris, B. Chen, J. Jahnotti, and M. F. Kasshoek, “The Click Modular Router”, *ACM Transaction on Computer Systems*, Vol. 18, No. 3, 2000, pp. 263–297.
[12] Linux-VServer Project, http://www.Linux-VServer.org.
[13] OpenVZ Project, http://www.openvz.org.
[14] OProfile, http://oprofile.sourceforge.net.
[15] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, “A Blueprint for Introducing Disruptive Technology into the Internet”, in *ACM SIGCOMM Computer Communication Review*, Vol. 33, No. 1, January 2003, pp. 59–64.
[16] S. Ratnasamy, S. Shenker, and S. McCanne, “Towards an Evolvable Internet Architecture”, in *Proc. ACM SIGCOMM ’05*, New York, NY, USA, 2005.
[17] E. Rosen and Y. Rekhter, “BGP/MPLS VPNs”, RFC 2547, IETF, March 1999.
[18] J. Touch and S. Hotz, “The X-bone”, in *Proc. 3rd Global Internet Mini-Conference at Globecom ’98*, Sydney, Australia, November 1998
[19] A. Whitaker, M. Shaw, and S. D. Gribble, “Scale and Performance in the Denali Isolation Kernel”, in *Proc. 5th Symposium on Operating System Design and Implementation, Boston*, MA, USA, December 2002.
[20] Y. Zu, R. Zhang-Shen, S.Rangarajan, and J. Rexford, “Cabernet: Connectivity Architecture for Better Network Services”, in *Proc. ACM ReArch ’08*, Madrid, Spain, December 2008.