Network Virtualization Architecture: Proposal and Initial Prototype

Gregor Schaffrath[‡] Christoph Werle[§] Panagiotis Papadimitriou[⋆]
Anja Feldmann[‡] Roland Bless[§] Adam Greenhalgh[†] Andreas Wundsam[‡]
Mario Kind[‡] Olaf Maennel[‡] Laurent Mathy[⋆]

[‡]Deutsche Telekom Laboratories / TU-Berlin, Germany
[§]University of Karlsruhe, Germany
[⋆]Computing Dept., Lancaster University, UK
[†]Dept. of Computer Science, University College London, UK

ABSTRACT

The tussle between reliability and functionality of the Internet is firmly biased on the side of reliability. New enabling technologies fail to achieve traction across the majority of ISPs. We believe that the greatest challenge is not in finding solutions and improvements to the Internet's many problems, but in how to actually deploy those solutions and re-balance the tussle between reliability and functionality. Network virtualization provides a promising approach to enable the co-existence of innovation and reliability. We describe a network virtualization architecture as a technology for enabling Internet innovation. This architecture is motivated from both business and technical perspectives and comprises four main players. In order to gain insight about its viability, we also evaluate some of its components based on experimental results from a prototype implementation.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: [Network Architecture and Design]

General Terms

Design, Management

Keywords

Network virtualization, Network architecture, Socio-Economics

1. INTRODUCTION

The Internet has been playing a central and crucial role in our society. Indeed, as the main enabler of our communications and information era, it plays a critical role in the work and business, the education, entertainment, and even the social life of many people. However, the Internet is also a victim of its own success as its size and scope render the introduction and deployment of new network technologies and services very difficult. In fact, the Internet can be considered to be suffering from "ossification" [11], a condition where technical and technological innovation meets natural resistance, as exemplified by the lack of wide deployment of inter-domain multicast or IPv6 in the public Internet. More precisely, while the network itself has indeed evolved tremendously in terms of size, speed, new sub-IP link technologies, and new applications, it is the architecture of the public Internet that has mostly remained the same and is difficult to change, because of the sheer size of the system.

The Internet, which aptly fulfills its current mission as a packet network delivering connectivity service, was also designed with assumptions that no longer describe future communications needs. Stronger security, better mobility support, more flexible routing, enhanced reliability, and robust service guarantees are only examples of areas where innovation is needed [15].

A promising approach to enable innovation is network virtualization, whereby several network instances can co-exist on a common physical network infrastructure. The type of network virtualization needed is not to be confused with current technologies such as Virtual Private Networks (VPNs), which merely provide traffic isolation: full administrative control, as well as potentially full customization of the virtual networks (VNets) are required. This also means that non-IP networks could then run alongside the current Internet realized as one future virtual network. The point is that each of these virtual networks can be built according to different design criteria and operated as service-tailored networks.

In this paper, we present a network virtualization architecture for a Future Internet, which we motivate by analyzing business roles. In contrast to the GENI initiative [16], our goal is not to provide an experimental infrastructure but to see which players are necessary to offer virtual network based services to everyone. We identify four main players/roles, namely the Physical Infrastructure Providers (PIPs), Virtual Network Providers (VNPs), Virtual Network Operators (VNOs) and Service Providers (SPs). This re-enforces and further develops the separation of infrastructure provider and Internet service provider advocated in [14, 20]. Indeed, we will show that the architecture encompasses other proposed network virtualization architectures.

This virtual network architecture is specifically designed to enable resource sharing among the various stakeholders, thereby increasing its adoptability. Indeed, in today's Internet, ISPs as well as service providers (e.g., Google) are continuously searching for opportunities to either increase revenue or to reduce costs by launch-

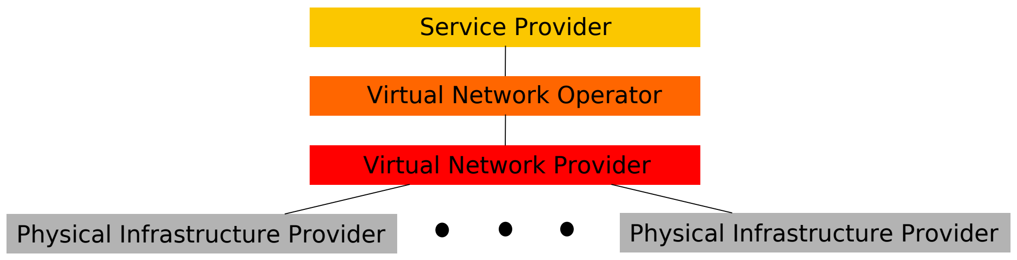


Figure 1: VNet Management and Business Roles

ing new services, investing in new technology (CAPEX) or by decreasing operational costs (OPEX). To understand the order of magnitude of the investment cost, consider that AT&T plans to invest 17–18 Bn \$ in 2009 [2] compared to a revenue of 124 Bn \$ in 2008 [3] and Deutsche Telekom invested 8.7 Bn € compared to revenues of 62 Bn € in 2008 [1]. Thanks to increased resource sharing, even a modest reduction in the investments of, say, 1% can result in several millions of savings per year.

The analysis of business roles is presented in Section 2. Section 3 provides details of the virtual network architecture, while a description of a prototype implementation is given in Section 4. In Section 5 we discuss our experimental results, while Section 6 provides an overview of related work. Finally, we conclude with an outlook in Section 7.

2. VIRTUALIZATION BUSINESS ROLES

The major actors in current Internet are service providers (e.g., Google) and Internet Service Providers (ISPs). Hereby, an ISP offers customers access to the Internet by relying on its own infrastructure, by renting infrastructure from someone, or by any combination of the two. Service providers offer services on the Internet. In essence, ISPs provide a connectivity service, very often on their own infrastructure, even if they also lease part of that infrastructure to other ISPs. For example, AT&T and Deutsche Telekom are mainly ISPs while Google and Blizzard are SPs.

Despite this “dual-actor landscape” [14, 20], there are already three main business roles at play in the current Internet: The (Physical) Infrastructure Provider (PIP), which owns and manages an underlaying physical infrastructure (called “substrate”); the connectivity provider, which provides bit-pipes and end-to-end connectivity to end-users; and the service provider, which offers application, data and content services to end-users.

However, the distinction between these roles has often been hidden inside a single company. For example, the division inside an ISP that is responsible for day-to-day operation of the network is rarely the one that is planning and specifying the evolution of the network.

By identifying these players, we can on the one hand identify different business opportunities and on the other hand disentangle the technical issues from the business decisions.

When considering the kind of network virtualization that enables the concurrent existence of several, potentially service-tailored, networks, a new level of indirection and abstraction is introduced, which leads to the re-definition of existing, and addition of new, business roles:

- Physical Infrastructure Provider (PIP), which owns and manages the physical infrastructure (the substrate), and provides wholesale of raw bit and processing services (i.e., slices), which support network virtualization.
- Virtual Network Provider (VNP), which is responsible for assembling virtual resources from one or multiple PIPs into a virtual topology.
- Virtual Network Operator (VNO), which is responsible for the installation and operation of a VNet over the virtual topology provided by the VNP according to the needs of the SP, and thus realizes a tailored connectivity service.
- Service Provider (SP), which uses the virtual network to offer his service. This can be a value-added service and then the SP acts as a application service provider, or a transport service with the SP acting as a network service provider.

These various business roles lead to the architectural entities and organization depicted in Figure 1.

In principle, a single company can fill multiple roles at the same time. For example it can be PIP and VNP, or VNP and VNO, or even PIP, VNP, and VNO. However, we decided to separate the roles as it requires different groups within the company. For example running an infrastructure is fundamentally different from negotiating contracts with PIPs about substrate slices. This in turn is fundamentally different from operating a specific network, e.g., an IP network for a service provider, which is the task of the VNO. As such splitting the roles increases our flexibility in terms of identifying the players, the corporate enterprises, that have that role. Thereby, it keeps the economic tussles away from the technical domain.

Note that both, a PIP as well as the VNP, deliver a virtualized network. Therefore, a VNP can act as a PIP to another VNP. However, one has to keep in mind that a VNP in contrast to a pure PIP has the ability to negotiate contracts with other PIPs and to assemble networks.

2.1 Business scenarios

Let us consider some of the new opportunities enabled by our separation of business actors (Figure 1) both for existing business entities and new players. For example, players A, B, and C may position themselves anywhere between and PIP and SP, e.g., A can operate only as a PIP. Then he acts as bit pipe ISP. On the other side of the spectrum C may decide to focus on an application service and outsource all other operational aspects. The business entity B may offer such an outsourcing service to A that encompasses VNO and VNP service buying its bit pipe from A.

Another possible position of A, B, and C along the possible spectrum is that A operates its own PIP and acts as VNP that assembles a VNet consisting of parts of its own physical infrastructure and from other PIPs. B may then offer the VNO service to the SP C. Yet another option is that C acts as SP, VNO, and VNP, acquiring resources from PIPs A and B.

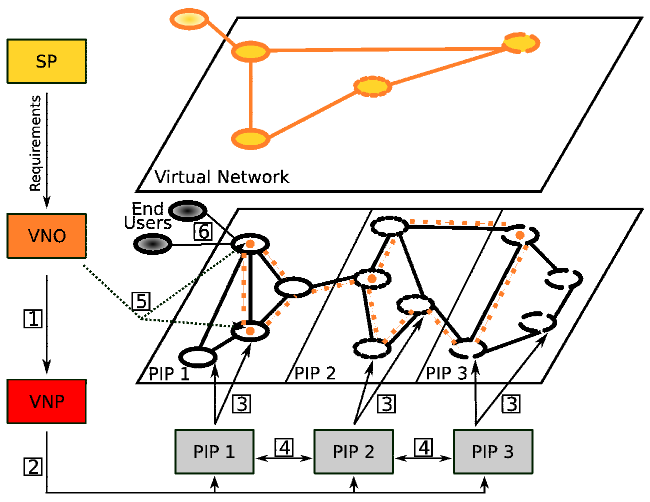


Figure 2: Interfaces between players

Lets consider how the proposed GENI architecture [16] fits within our framework. The GENI clearinghouse is a VNP. The experimenter is the VNO and if they desire the SP. As such GENI also realizes the split between PIP and VNP. However, as GENI does not yet consider federation it does not consider the implications of having to handle multiple PIPs.

3. VNET CONTROL ARCHITECTURE

In this section, we introduce our *VNet Control Plane Architecture* which provides the control and management functions for the virtual network architecture to the various actors. The control plane must perform a balancing act between the following tussles:

- Information disclosure against information hiding.
- Centralization of configuration and control against delegation of configuration and control.

The information disclosure tussle is a subtle one and we will try to illustrate this through a couple of scenarios. The first scenario is accounting where each customer needs to be able to satisfy themselves that they are getting their contractual requirements without the provider releasing sensitive information to them or others. For example, a customer can request certain Quality of Service (QoS) guarantees across the PIP's network without any information on the exact physical path. The second scenario, which is arguably the most challenging is network debugging. It is a complex problem to provide enough information to, for example, a VNO to enable them to debug a problem with the physical path in PIP without providing too much information.

Where configuration and control are undertaken is another tussle. For example, the PIP should be able to render/delegate low level management of the virtualized network components via the VNP to a VNO, whilst hiding it from another VNO.

3.1 Control Interfaces

In the following we identify the control interfaces (see Figure 2) in our architecture by discussing how the various players interact in order to setup a VNet.

To begin with, the SP hands the VNO his requirements. Then the VNO needs to add his requirements and any constraints he imposes on the VNet. This description is subsequently provided (via *Interface 1*) to the VNP of his choice, which is in charge of assembling the VNet. The VNP may split the request among several PIPs, e.g., by using knowledge about their geographic footprints, and send parts of the overall description to the selected PIPs (via *Interface 2*). This negotiation may require multiple steps. Finally, the VNP decides which resources to use from which PIP and instructs the PIPs to set up their part, i.e., virtual nodes and virtual links, of the VNet (*Interface 3*). Now, all parts of the VNet are instantiated within each PIP but they may still have to be interconnected (*Interface 4*). The setup of virtual links between PIPs—in contrast to Interface 3—needs to be standardized in order to allow for interoperability across PIP domains. Once the whole VNet has been assembled, the VNO is given access to it (*Interface 5*). This interface is also called "Out-of-VNet" access and is necessary as, at this point in time, the virtual network itself is not yet in operation. Thus, a management interface outside of the virtual network is needed. Once the virtual network has been fully configured by the VNO and the service is running, end-users can connect to the virtual network (*Interface 6*).

We now discuss how each player benefits from this virtualization architecture: PIPs can better account for the constraints imposed by the VNets. For example, before scheduling maintenance work or for traffic engineering purposes, they might migrate some virtual nodes to minimize downtime or to optimize their traffic flow. This is possible as long as the the new location is embedding-equivalent, i.e., satisfies all of the requirements and imposed constraints, and enabled by the level of indirection introduced by our architecture and the use of modern migration mechanisms[18, 10]. For VNPs, migration between PIPs offers a mechanism to optimize their revenue by choosing competitive and reliable PIPs. As pointed out by [20], the removal of the requirement for individual negotiations between VNOs and all participating PIPs facilitates the entry of new players into the market. Furthermore, SPs may outsource non service specific network operation tasks to other entities and thereby concentrate on their core interests relating to their respective business model. Migration processes are transparent to the VNOs. Note that they cannot trigger migration directly; however, by altering their requirements, VNOs may indirectly initiate resource migration.

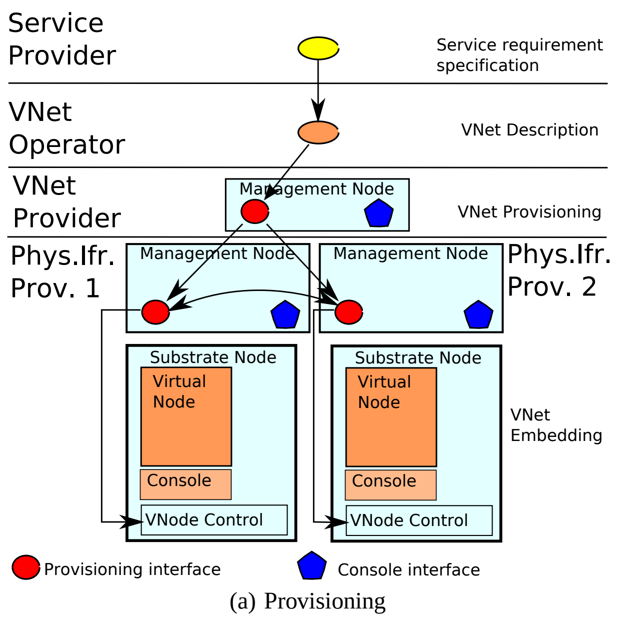
3.2 VNet Instantiation

Setting up a VNet, see Figure 3 (a), starts from a VNet specification. For this we need resource description languages for both the VNet topology, including link/node properties, as well as service level requirements. These description languages should neither be too constrained – to allow the VNP and the PIPs freedom for optimizations – nor too vague – to enable a precise specification. Therefore it is out of scope for this paper.

To setup the VNet each player, for its domain, has to: formulate resource requirements, discover potential resources and partners, negotiate with this partners based on VNet resource description, and construct the topology.

SP: The SP specifies his service specific requirements which might include a VNet topology. In addition, he may specify the kind of interface it needs for service deployment and maintenance, e.g., what level of console access. He then delegates the instantiation to the VNO of its choice. Once the VNet is instantiated the SP deploys his service using the interface provided by the VNO.

VNO: The VNO uses the specification it receives from the SP and generates a VNet specification. It then negotiates with various VNPs on the basis of the VNet specification. Once a VNP is selected the VNO has to wait for the VNP to assem-



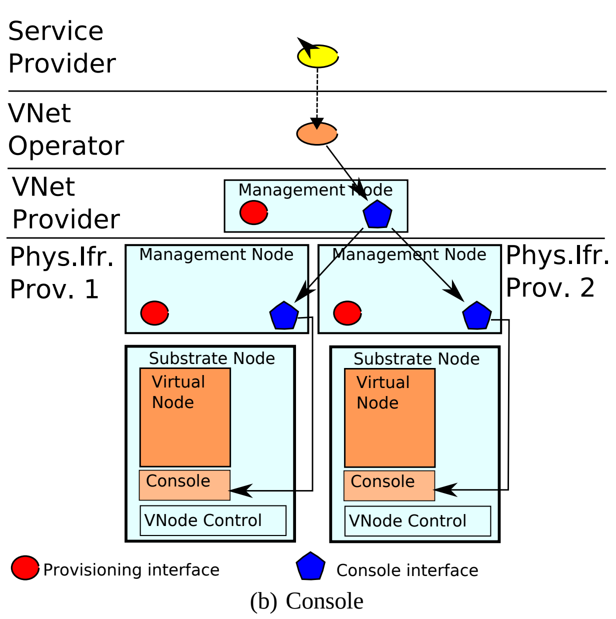


Figure 3: VNet provisioning (a) and console architecture (b).

ble the VNet. When it has access to the VNet which consists of a data and a control network it can use the control network, also referred to as the management console interface or out-of-VNet access, see Section 3.3, to instantiate the network service. Finally, it may instantiate the control interface needed by the SP.

VNP: Upon receiving the VNet resource description, the VNP identifies candidate PIPs and splits the VNet resource description into multiple subsets. It then negotiates with the candidate PIPs regarding the necessary substrate resources. Once the PIPs have assembled the pieces of the VNet it completes it. Finally, it provides a management console access for the whole VNet by relying on the management interfaces of the PIPs. Note, that a VNP may aggregate requests for multiple VNets. It may also request additional resources from the PIPs to satisfy future requests. In this sense a VNP can act as any reseller would.

PIP: Based on the VNet topology descriptions a PIP receives it identifies the appropriate substrate resources and allocates them. The PIP has the ability to migrate other VNets in order to free resources for new requests. After setting up the VNet on its substrate he returns both the data and the control part of the VNet. The control part includes the PIP level management consoles to allow the configuration of the virtual nodes. Since VNets may span across multiple PIPs some virtual links may have to be setup across PIPs.

3.3 Out-of-VNet Access

Each VNet consists of two planes: a data plane and a control plane. The data plane is what one commonly refers to in the context of virtual networks. But, per default, any VNet after instantiation is just an empty set of resources that have to be configured. Moreover, a control plane is necessary during VNet operation for specific VNet management tasks. Such a management access architecture is shown in Figure 3 (b). Every player maintains a control interface hosted on dedicated management nodes for “out-of-VNet” access to the VNet resources. This interface provides a console interface as well as control options such as virtual resource power cycling. As there are multiple levels of indirection, it also stores the location of the subsequent control interfaces. Control requests are thus relayed from player to player until they get to the appropriate virtual substrate component. In addition, to enable migration both within the PIP as well as across PIPs, consistent interface updates have to be supported.

Typical interfaces for out-of-VNet access are consoles, either via tty or a graphical interface. Requests for such interactive and maybe traffic intensive connections can be handled via proxy chains. Such a proxy chain can be setup by relaying requests from one player to another while instantiating appropriate connection proxies and returning their access information. To render migration seamless, connection ruptures should be hidden, e.g., by keeping the reestablishment time of the proxy chain short or by proactively moving the connections.

3.4 End-user/End-system access to VNets

End-users/End-systems wishing to dynamically join a VNet need to be authenticated at their physical attachment point in their local PIP before being connected to the VNet. The authentication occurs via an authentication channel, which may serve one or more virtual networks, provided by their local PIP, see Figure 4. For this purpose collaborating PIPs may participate in a provisioning and management virtual network. The attachment process for an end-system can then be summarized as follows.

- The end-system accesses the authentication channel via any local access mechanisms imposed by the local PIP.
- The end-system requests access to the VNet via the local authentication portal which validates the system locally or relays the request across the Provisioning and Management network to the corresponding VNO authentication portal.
- The authorized end-system is then connected to the requested VNet and is able to configure its address, routing, and network service inside the VNet.

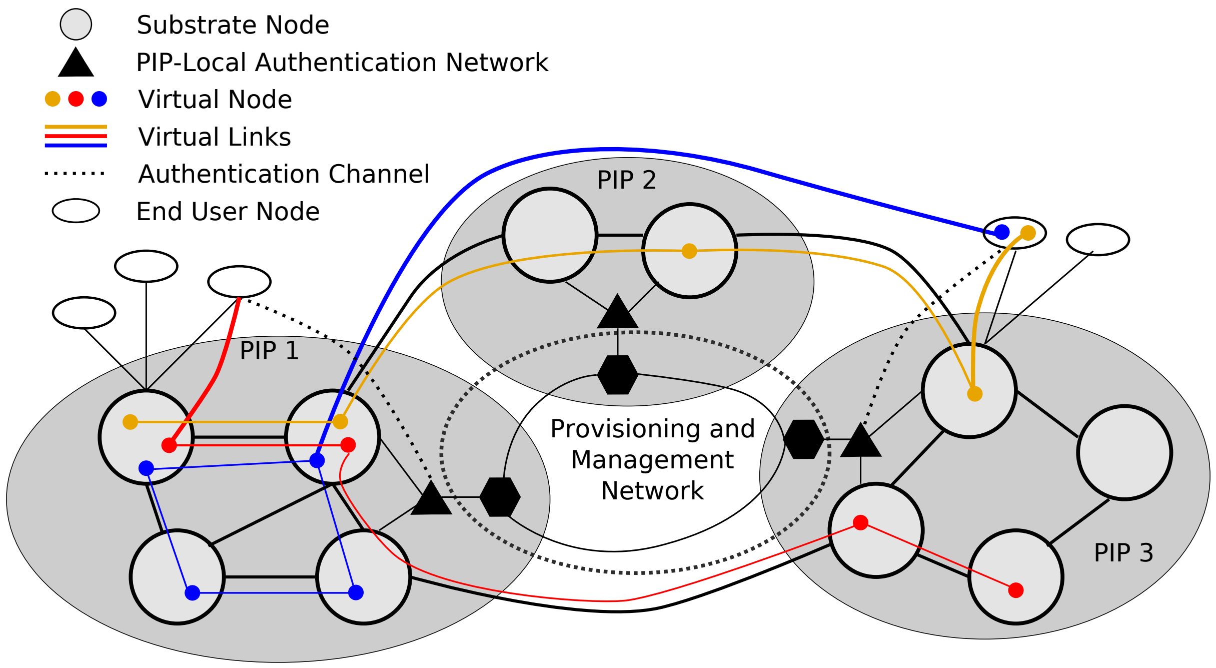


Figure 4: End-System Attachment

4. PROTOTYPE IMPLEMENTATIONS

In this section, we briefly describe two aspects of our prototype implementations for the VNet architecture.

4.1 Virtual Network Instantiation

Our prototype of network instantiation is realized in the context of *Heterogeneous Experimental Network*[1] (HEN) [4]. The prototype takes advantage of node and link virtualization technologies to allow the instantiation of VNets on top of a shared substrate. Specifically, we use Xen's paravirtualization for hosting virtual machines on substrate nodes. The virtual machine monitor (also referred to as hypervisor) schedules CPU accesses to the separate guest domains and provides an adequate level of isolation and high performance [13].

We use a XML schema for the description of virtual resources with separate specifications for nodes and links. The prototype currently considers a single PIP which controls a fixed number of HEN physical nodes. The VNP directly communicates with the management node of the PIP. This node is responsible for realizing all the VNet requests to the PIP. After resource discovery, the PIP management node signals individual requests to the necessary substrate nodes. Each substrate node handles the request within their management domain (Dom0) which then creates the necessary virtual machines as guest domains (DomUs).

For the inter-connection of the virtual nodes, we currently use IP-in-IP encapsulation tunnels. The topologies are built by VLAN configuration on the HEN switch. This process is automated via a *switch-daemon* which receives VLAN requests and configures the switch accordingly. Virtual links are established by encapsulating and demultiplexing packets, as shown in Figure 5. More precisely, each virtual node uses its virtual interface to transmit packets, which are captured by Click [17] for encapsulation, before being injected to the tunnel. On the receiving host, Click demultiplexes the incoming packets delivering them to the appropriate virtual machine. For packet forwarding, we use Click SMP with a polling driver. In all cases, Click runs in kernel space. Substrate nodes that route packets consolidate all forwarding paths in a single domain (Dom0) preventing costly context switches; hence, packet forwarding rates are very high [13].

[1]HEN comprises over 110 computers connected together by a single non-blocking, constant-latency Gigabit Ethernet switch.

4.2 Out-of-VNet Access

The prototype implementation of the management console has been built using standard UNIX tools, with a database holding the mappings from virtual resource identifiers to management and substrate nodes along with user credentials. A simple proxy at the management node facilitates the forwarding of graphical (VNC) and textual (serial) consoles towards the corresponding target device identified through a database lookup once the connection request has been authorized. The connections are chained through the chain of management nodes, for example, the VNP management node relays request to PIP management node. The proxy connection is terminated by either a disconnection from either the client or server. The prototype supports recursive extension of the management architecture at the VNP level. This allows both for VNPs acting as PIPs and reselling their resources to other VNPs, as well as internal structuring of an ISP.

The control interface provides virtual machine control enabling the migration of virtual nodes between PIPs when instructed from the VNP. The standard virtual machine controls of 'start', 'stop', 'reboot', 'suspend', and 'resume' are supported from the VNO.

5. RESULTS AND DISCUSSION

In this section, we report on some early results from our prototype implementation to support the feasibility of the proposed virtual network architecture.

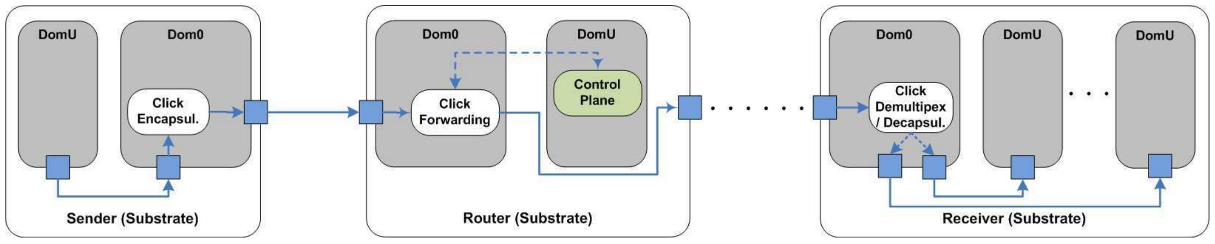


Figure 5: A virtual link on HEN prototype

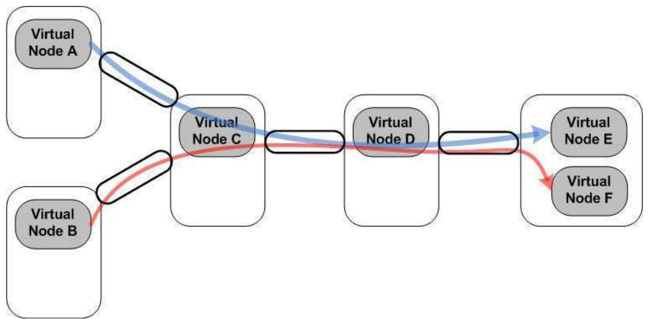


Figure 6: Experimental topology.

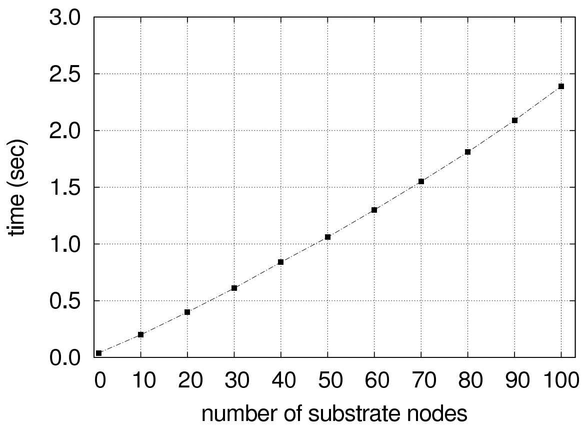


Figure 7: Average time required to update resource information for all substrate nodes.

5.1 Virtual Network Instantiation

We use tests with our virtual network instantiation prototype to show: (i) that VNet instantiation times within a PIP are reasonable, and (ii) to explore the management overhead, in terms of CPU load for the VNP and the substrate nodes.

To test VNet instantiation times we use the VNet topology shown in Figure 6. The experiments are conducted in the HEN testlab on Dell PowerEdge 2950 servers, each with two 2.66 GHz quad-core Intel X5355 and 8 Gigabit interfaces. The instantiation time includes the exchange of resource information, i.e., the VNet description, between VNO, VNP and the PIP, resource discovery within the PIP, VNet creation and configuration of all virtual machines, setup of the tunnels, and enabling management access to the virtual nodes. With on-demand creation and booting of the virtual machines, it takes 109.5 seconds on average across 20 tries with a small standard deviation of 4.4 to instantiate the virtual network. Alternatively, if the PIP can allocate physical resources to virtual machines in advance (so that booting them is not required upon receiving a VNet request), the instantiation time drops to 16.8 seconds with a standard deviation of 0.4. As such our prototype is able to quickly provision VNets that are ready to be operated and managed by the VNO.

A time consuming step, besides virtual machine creation, is resource discovery within the PIP. Within the prototype implementation, it requires the exchange of available resources (per host) among the PIP management node and each one of the substrate nodes that meet the criteria defined with the VNet request (e.g., location). Figure 7 shows how the average time for updating resource information scales with the number of substrate nodes in HEN. It scales linearly. As such one either has to increase the number of management nodes as the PIP scales or establish a network-wide authoritative configuration manager that then delegates the instantiation of the individual nodes across multiple configurators.

We also used OProfile [5] to monitor the CPU utilization for all participating nodes during VNet instantiation, which is shown in Table 1. While these results are specific to our implementation they show that VNet instantiation does not impose an unreasonable overhead.

Table 1: %CPU during VNet instantiation

	min	avg	max	stddev
VNO	10.0	12.5	14.7	1.6
VNP	20.4	23.5	25.5	1.5
Substrate Node	16.3	19.2	22.6	2.1

Even though we use a single PIP throughout our experiments, we do not expect significant additional overheads when moving to multiple PIPs as the resource discovery can proceed in parallel. In the current setup using multiple PIPs should only slightly increased the CPU load for the VNP. The only additional overhead is the communication with multiple entities and the assembly of the VNet from the participating PIPs.

5.2 Out-of-VNet Access

We use test with our Out-of-VNet access prototype to address the question of scalability and the feasibility of migration of the underlying nodes without breaking the management connection.

In principle, scalability concerns arise with regards to the number of managed virtual resource instances, the number of role instances (PIPs, VNPs, VNOs), and the depth of the management node structure. The performance of the first two is very implementation dependent. Therefore we decided against evaluating these. The current implementation of our prototype is not optimized for

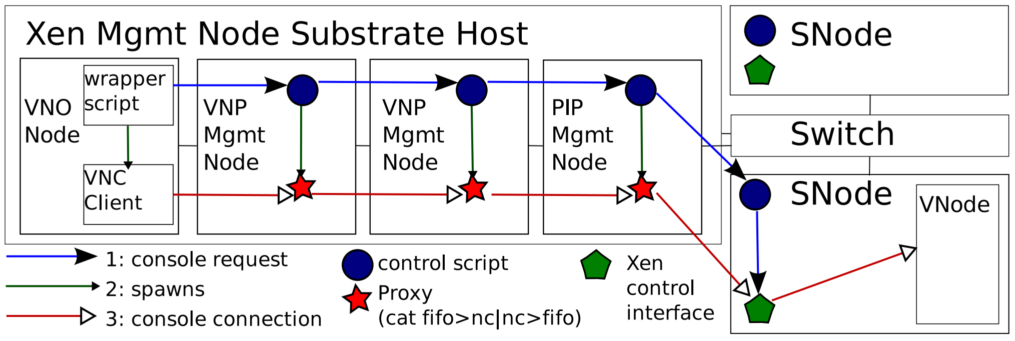


Figure 8: Console proxy measurement setup

performance but for features. Nevertheless, we expect good scalability as high-performance databases exist and load-balancing can be used to distribute the proxy load across multiple physical servers.

However, we study the impact of the depth of the management node structure on console connection establishment and migration latencies. These latencies have two components: network communication and processing overhead at the management nodes (assuming no request buffering). Given that the network communication latency is unavoidable and dominated by the location of the management nodes and the customer/resource assignment strategies we focus on the processing overhead. Based upon the assumption that a deterministic overhead is introduced for each node type we measure the overhead on a per-management node type.

The chain length in the experiments is 4 rather than 3. We included a VNP which acts as PIP, see Figure 8. A VNC client is connected to a VNode then migrated between the two substrate nodes (SNodes). A wrapper script requests a new console after every migration and measures the time required until it receives the VNP level proxy access information. Then it restarts the client. Every node in the chain measures the execution time of the management script (including all operations, lookups, and network communication) until its termination upon transmission of the proxy access information. Durations are measured using the Unix 'time' tool. The resulting processing time (User+System) and elapsed real time are shown in Figure 9.

The experimental setup consists of 5 machines: 1 dual Intel Core2 Duo with 2GB RAM hosting 7 paravirtualized management nodes, and 4 dual Dual-Core AMD Opteron machines with 2GB RAM which act as substrate nodes. The machines are interconnected via a Cisco C2960 switch.

Figure 9 shows that the CPU time required for script execution does not exceed 0.1s for intermediate nodes and 0.2s for the SNode. The added delay is about 0.02s per hop. This includes communication delay as well as the unaccounted delay due to inetd/telnetd operations. For most experiments it is less than 0.6s in total. Although measurement 14 shows a spike in real time delay at the SNode this coincides with an overall low processing time on the same node. We therefore assume that this spike is caused by OS process scheduling overhead.

Considering the minimal delays of 0.2s for SNodes and 0.1s for intermediate node our prototype easily scale to a depth of 8 without exceeding 1s in delay for proxy setup.

6. RELATED WORK

Over the last years virtual network architectures have been an area of active research. Some groups have focused on using network virtualization to enable larger-scale and more flexible testbeds. Other groups aim at virtualizing production networks or even the Internet.

6.1 Architectures for testbeds

Virtualization plays a key role in creating flexible testbeds for Future Internet research.

Planetlab family:

PlanetLab [7] is a highly successful example of a distributed, large scale testbed. PlanetLab has a hierarchical model of trust which is realized by *Planet Lab Central* (PLC). PLC is operated by the PlanetLab organization and is the ultimately trusted entity that authorizes access to the resources. Other actors are the *infrastructure owners* and the *users* that run their research experiments on planet lab. For each experiment virtual machines on various nodes are grouped to *slices* that can be managed and bootstrapped together. As the deployed virtualization mechanism offer only container based virtualization capabilities at the system level and do not virtualize the network stack PlanetLab offers no network virtualization as such.

VINI, as proposed by Bavier et al. [8], is a testbed platform that extends the concept of virtualization to the network infrastructure. In VINI routers are virtualized and interconnected by virtual links. As such VINI allows researchers to deploy and evaluate new network architectures with real routing software, traffic loads, and network events. VINI supports simultaneous experiments with arbitrary network topologies on a shared physical infrastructure. VINI builds on the architecture and management framework introduced by PlanetLab by extending the management with interfaces to configure virtual links. The first implementation based on *User Mode Linux* [19] however offers only limited performance.

An updated VINI platform, Trellis [9], allows for higher forwarding performance. It introduces a lower level system virtualization architecture that uses container based virtualization techniques for both system and network stack virtualization. Therefore virtualization flexibility is limited to the user space. VINI provides rudimentary concepts for end-user attachments [6] using OpenVPN tunnels and a single central gateway. Obviously, this solution would not scale to virtualization applied on an Internet wide scale.

Downloadable distributions of the Planetlab control framework and VINI are available as MyPLC and MyVINI, respectively.

Emulab: Emulab [12] also is a very popular testbed platform. Its offers a sophisticated management and life-cycle processes and does not offer that much of a network architecture. Emulab offers virtual topology configuration based on ns2 configuration files

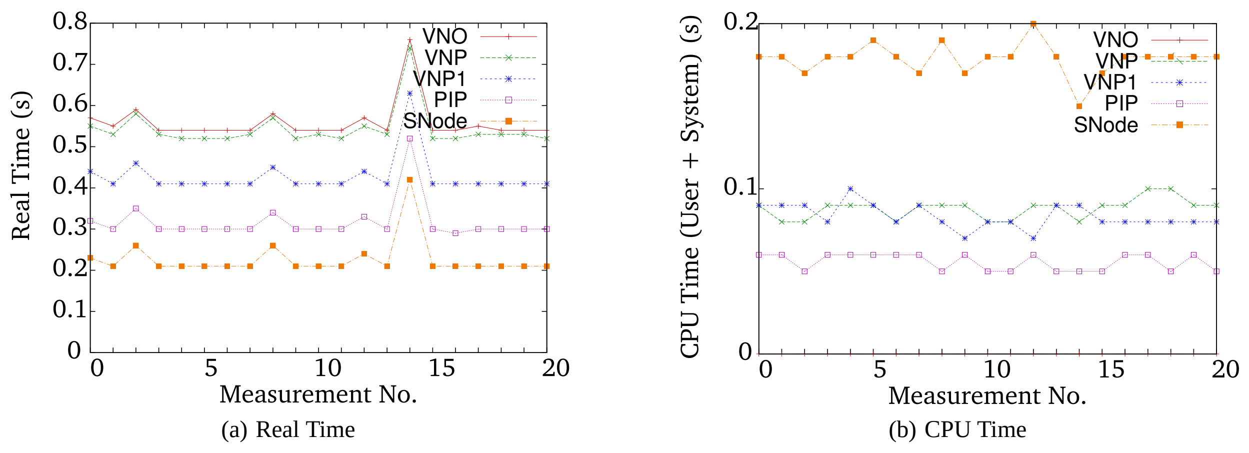


Figure 9: Console proxy setup time

and automatic bootstrapping of experiment nodes. Initially, Emulab focused on dedicated servers. Virtualization capabilities based on improved FreeBSD jails were added later.

GENI:

GENI [16] is a large-scale U.S. initiative for building a federated virtualized testbed aiming at providing a powerful virtualized testbed for experimental purposes. Here, all operations are signed off and managed by a central *Geni Clearing House*, which can thus be regarded as analogue to our VNP. As a possible growth path, GENI plans on supporting federated clearing houses, but its design has not yet been presented in detail. During phase 1 of the development both—VINI/Planetlab and Emulab—are used as GENI prototypes (*ProtoGeni*).

All testbed oriented architectures mentioned above do not consider several key factors relevant for virtualizing the (commercial) Internet: They assume a hierarchical trust model that centers on a universally trusted entity, e.g., the PLC/GENI clearinghouses. To overcome this limitation we consider competing players with individual administrative zones that have only limited trust and also have the desire to hide information, e.g., their topologies. Economic models and use cases are not critical for testbed designs but are crucial for the adoption of an Internet-wide virtualization architecture.

6.2 Architectures for Production Networks

CABO [14] proposes to speed up deployment of new protocols by allowing multiple concurrent virtualized networks in parallel. To this end, infrastructure providers are to manage the substrate resources while service providers are allowed to operate their own customized network inside the allocated slices. These slices are acquired by negotiations of service providers with a series of infrastructure providers.

This idea in refined by Cabernet [20] which introduces a “Connectivity Layer” between the above mentioned roles. This layer is responsible for the splicing of partial networks provided by the Infrastructure Layer and the presentation of a single network to the Service Layer. It facilitates the entry of new service providers by abstracting the negotiations with different infrastructure providers and allows for aggregation of several VNets into one set of infrastructure level resource reservations.

While this structure relates to our proposal, our approach differs as we propose to split the service provider and connectivity provider role into the three roles of VNP, VNO, and SP. These roles allow for a more granular splitting of responsibilities with respect to network provisioning, network operation, and service specific operations which may be mapped to different business entities according to various different business models. Furthermore, we extend the framework by considering privacy issues in federate virtual network operations and business aspects.

7. CONCLUSION AND OUTLOOK

We presented a VNET Control Architecture that comprises four main entities reflecting different business roles, namely the Physical Infrastructure Providers (PIPs), Virtual Network Providers (VNPs), Virtual Network Operators (VNOs), and Service Providers (SPs).

An important aspect of this control architecture is that it defines the relationships that govern the interaction between the players, *without* prescribing their internal organization, structure and policies. In other words, every entity manages its resources as it sees fit. This property is crucial to the viability of the proposal, as it ensures the protection of business interests and competitive advantages. Furthermore, our architecture encompasses other proposed network virtualization architectures, e.g., GENI and Cabernet [16, 20].

In support of this flexible resource management strategy, we emphasize the need for an Out-of-VNet access management interface to allow some basic control of virtual nodes from the outside of the VNet.

We implemented a simple prototype for the VNET control architecture, using simple virtual nodes realized with XEN and Click. Evaluation for VNet instantiation showed that it can scale linearly with the number of nodes. The second evaluation considered the Out-of-VNet access via a proxy hierarchy in order to get back to a virtual node after its migration in the substrate. It showed that this access can be restored in a reasonable short time frame to allow for even deeper proxy hierarchies if necessary.

These results are encouraging since they show the viability of parts of the architecture. In the future we plan to extend to prototypes to realize the full architecture and start experimenting with advanced optimization techniques to realize the possible gains in OPEX and CAPEX.

Acknowledgments

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

(FP7), and the Virtu project, funded by NTT DoCoMo Euro-Labs. We would like to thank our colleagues in both projects for many fruitful discussions.

8. REFERENCES

[1] Annual Report Deutsche Telekom AG, 2008. `http://www.download-telekom.de/dt/StaticPage/62/37/50/090227_DTAG_2008_Annual_report.pdf_623750.pdf`.

[2] AT&T Reports Fourth-Quarter and Full-Year Results. `http://www.att.com/gen/press-room?pid=4800&cdvn=news&newsarticleid=26597`.

[3] AT&T Reports Fourth-Quarter and Full-Year Results. `http://www.att.com/gen/press-room?pid=4800&cdvn=news&newsarticleid=26502`.

[4] Heterogeneous Experimental Network. `http://hen.cs.ucl.ac.uk`.

[5] OProfile. `http://oprofile.sourceforge.net`.

[6] VINI End User Attachment. `http://www.vini-veritas.net/documentation/pl-vini/user/clients`.

[7] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating system support for planetary-scale network services. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[8] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. In *Proceedings of SIGCOMM '06*, pages 3–14. ACM, 2006.

[9] S. Bhatia, M. Motiwala, W. Mühlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Trellis: A platform for building flexible, fast virtual networks on commodity hardware. In *ACM CoNEXT ROADS Workshop*, 2008.

[10] C. Clark, K. Fraser, S. H, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proceedings of ACM/USENIX NSDI '05*, pages 273–286, 2005.

[11] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in Cyberspace: Defining Tomorrow's Internet. In *Proceedings of SIGCOMM '02*, pages 347–356. ACM, 2002.

[12] J. Dike. A usermode port for the linux kernel. In *USENIX Linux Showcase & Conference*, 2000.

[13] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy. Towards High Performance Virtual Routers on Commodity Hardware. In *Proceedings of ACM CoNEXT '08*, Dec. 2008.

[14] N. Feamster, L. Gao, and J. Rexford. How to Lease the Internet in Your Spare Time. *SIGCOMM CCR*, 37(1):61–64, 2007.

[15] A. Feldmann. Internet clean-slate design: What and why? *SIGCOMM CCR*, 37(3):59–64, 2007.

[16] GENI: Global Environment for Network Innovations. http://www.geni.net.

[17] E. Kohler, R. Morris, B. Chen, J. Jahnotti, and M. F. Kaashoek. The Click Modular Router. *ACM Transaction on Computer Systems*, 18(3):263–297, 2000.

[18] 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. *SIGCOMM CCR*, 38(4):231–242, 2008.

[19] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[20] Y. Zhu, R. Zhang-Shen, S. Rangarajan, and J. Rexford. Cabernet: Connectivity Architecture for Better Network Services. In *Proceedings of ReArch '08*. ACM, 2008.