<http://kth.diva-portal.org>

This is an author produced version of a paper published in *2012 IEEE International Conference on Communications (ICC)*.

This paper has been peer-reviewed but does not include the final publisher proof-corrections or proceedings pagination.

© 2012 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Citation for the published paper:

Turull, Daniel, Markus Hidell, Peter Sjödin.
libNetVirt: the network virtualization library
2012 IEEE International Conference on Communications (ICC).
Access to the published version may require subscription.
Published with permission from: IEEE

libNetVirt: the network virtualization library

Daniel Turull, Markus Hidell, Peter Sjödin
KTH Royal Institute of Technology, School of ICT
Stockholm, Sweden
Email: {danieltt,mahidell,psj}@kth.se

Abstract—Network virtualization has been an important research topic for many years but still suffers from the lack of an abstraction level like the one present in virtualization of computing and storage. Our work in progress presented here proposes an architecture for such a network virtualization abstraction. It is deployed as a library, similar to libvirt in computer virtualization, with a unified interface towards the underlying network specific drivers. The architecture will allow management tools to be independent of the underlying technologies. In addition, it will enable programmatic and on-demand creation of virtual networks. A common set of calls is defined to instantiate different virtual networks, using a single node view to provide the user with a suitable abstraction of the network. We describe a prototype of our proposed architecture on top of an OpenFlow-enabled network. We demonstrate its feasibility for creating isolated virtual networks in a programmatic and on demand fashion.

I. INTRODUCTION

The abstraction of underlying technologies offers flexibility in the operations on the physical equipment, and is a common approach today in modern systems. An example is machine virtualization where computers are represented in a straightforward fashion as CPU, memory, storage, and network interfaces. Another example is disk virtualization where logical volumes are presented instead of physical disks. Both types of virtualization are fairly well understood and agreed upon. The common denominator here is that the logical view is decoupled from the physical hardware, which allows resources to be shared.

However, virtualizing networks is a different and more complex challenge. Networks are still coupled with the physical infrastructure [1]. Network virtualization is a mechanism for sharing a physical network. Virtual networks have been around for several years but nowadays they are one of the main focuses in the networking community. There are different technologies and approaches that can be used to share a physical network or substrate, such as, VLANs, Virtual Private Networks, virtual routers or programmable virtual routers.

Most of these technologies require a significant degree of manual configuration to be done by the network operator, something which makes the adoption to dynamic environments quite impractical. In different projects, efforts have been made to generate frameworks [2]–[5] to manage the creation and removal of point-to-point connections but there is not a unique abstract network virtualization interface that is independent of the different underlying technologies.

Our view is that a unified framework for network virtualization is necessary to expand network functionalities and

to provide a single network view. This will make possible to share network resources and to reduce costs. We believe that an easy way to create networks on demand, programmatically and with Quality of Service (QoS) is required.

The purpose of the work presented here is to establish a common set of network abstractions to allow users to easily provide network resources programmatically in a similar way to how machine virtualization is done. Our goal is to define a simple view of the network, as it has been done with libvirt [6] in the context of machine virtualization. Libvirt is a library which abstracts the description of a virtual machine from the underlying hypervisor. Libvirt can be used for other applications and management tools to control the creation, modification and removal of virtual machines.

In previous work, others have focused on the creation of complete networks graphs with virtual links and routers on top of a physical infrastructure [7]–[11]. These require a complete description of the desired topology and elements, which makes the management of the virtual infrastructure unnecessarily complex. Another alternative, proposed by Keller and Rexford [12], is to present the network as a single router, where all in-network functionality is covered. It has advantages for both players: users and network providers. Users do not need to manage the physical network to run their services and network providers can offer their platform with an added value.

The work in progress described in this paper is referred to as *libNetVirt*, and it uses the single router abstraction to describe a network. It is composed by a set of drivers. Each driver implements the required configuration for a specific underlying technology. LibNetVirt aims to reduce the time for the creation and termination of virtual networks. A common set of calls is defined and used to instantiate different virtual networks, in a programmatic and on demand fashion.

We have implemented an open source [13] prototype of our proposal on top of an OpenFlow-enabled [14] network. We describe the internal architecture to make possible to create virtual networks using OpenFlow. We have chosen OpenFlow because it offers a programmatic interface to remotely control the forwarding elements in the network.

The rest of the paper is organized as follows: Section II introduces the library libNetVirt, which uses the network abstraction. Section III explores the design and implementation of a libNetVirt driver for an OpenFlow-based network. Section IV describes a use case of the library in cloud networking. Section V discusses the related work. Finally, in Section VI we summarize the work.

Fig. 1. LibNetVirt architecture

II. LIBNETVIRT

We have designed a C library with Python wrappers, which provides an abstraction for the network where the minimum requirement for the user is to provide the endpoints to interconnect. This section describes the network view that we are using, the main elements of the architecture and the basic operations.

The target of the library is to be used in any network where virtualization is required. The operations are called from upper layers of the management platform. However, we decided to keep security policies outside the library in order to keep the design simple and to remove complexity from the library core. Upper layers will be responsible for security aspects.

A. Virtual Network View

Traditionally, the approach taken when dealing with Virtual Networks (VN) is to map virtual nodes and links (vertices and edges) to physical nodes and links. This provide full management capabilities to the users [12], providing a complete graph of the network view. However, users would need to manage the virtual network in the same way as a regular physical network would be managed, for instance, providing traffic engineering or coping with link failure. We believe that users that want to manage their application in dynamic environments do not need the added complexity of managing a complete network. The single router abstraction, addresses this problem defining the network as a unique node where all the functionalities are inside it. All actions are defined in this single element, where all the complexity is hidden from the user and it provides to him an easy way to interact with network resources.

B. Architecture

LibNetVirt is separated in two different parts: generic interface and drivers (See Fig. 1). The generic interface is a set of functions that allow interacting with the VN and executing the operations in the specific driver. A driver is a technology dependent element which communicates with the required components to manipulate the VN in the physical equipment. This modularity allows us to change the underlying technology without affecting the management tools. Another alternative is to use independent controllers for each platform, but, in this case, reusing common functionalities is not possible.

The management application uses the calls from libNetVirt to operate the network. It can provide the description of it with an XML file. LibNetVirt parses the file description and invokes the specific function of the correct driver. In our first approach,

we have identified the following parameters that can describe a VN: *endpoint*, *forwarding* and *path constraint*. However, it is possible to further extend these definitions and define new endpoint types.

- **Endpoint:** is the termination of a VN. A resource is connected to an endpoint. An endpoint is described with multiple fields, where some are mandatory and others optional. The mandatory fields are:

- *uuid*: unique identifier for the endpoint in the VN. Used to identify the endpoint.
- *switch id*: identification of the physical switch.
- *port*: edge port, where the resource is connected.

In addition, optional fields can be used to distinguish traffic from different VNs that use a shared port:

- *mpls*: MPLS tag for the incoming/outgoing traffic. Also can be used to interconnect different networks.
- *vlan*: the same function of *mpls* but for VLAN tags.
- Other flow fields can be added in order to extend the definition of a VN.
- Traffic limitations, such as, maximal input bandwidth, can also be added in the endpoint definition.

- **Forwarding:** defines the type of packet forwarding. We can have forwarding based on L2 (based on MAC addresses) or L3 (based on IP addresses). It can be extended with new forwarding types.
- **Path constraints:** defines unidirectional constraints between two endpoints. A constraint can be any QoS specification required in the VN, such as, minimal bandwidth between two endpoints. It contains the *uuid* for the source and destination endpoints and the QoS constraint.

C. Basic Operations

A set of basic operations is required to manage a VN. These operations can be executed programmatically from the user interface or manually by a network operator. The programmatic executions are called from upper layers of the management platform. We have identified the following basic operations, which might be further expanded.

- Creation of a VN: the user defines the desired VN in an XML file or through the libNetVirt API.
- Removal of a VN: the user needs to provide the *uuid* of the VN that wants to remove.
- Addition/Modification/Removal of an endpoint: the user adds the description of the endpoints that wants to add or remove.
- Addition/Modification/Removal of a path constraint, in order to manipulate on demand QoS constraints.

III. OPENFLOW DRIVER

We decided to implement a libNetVirt driver for OpenFlow. OpenFlow [14] is a novel architecture which allows decoupling of the data plane from the control plane. Basically, it is a protocol which allows the communication between a remote controller (control plane) and a switch (data plane). The packet forwarding is based on flows and it can be customized to

any set of fields of a packet. The controller is in charge of installing the appropriate forwarding rules to the switch. When the switch receives a packet that does not match any rule, the default action is to forward the packet to the controller. The controller decides the action to be made and install the appropriate rules if it is necessary.

OpenFlow gives us the flexibility to control our network in a programmatic way and with a global view. In addition, it provides a remote control of the forwarding plane making easy to experiment with different forwarding strategies.

In order to implement drivers for other technology that lack the benefits of OpenFlow, it is necessary to create agents near the affected routers. These agents will communicate with the libNetVirt driver and they can use CLI scripts or use netconf [15] to change the router configuration.

A. OpenFlow options

There are several platforms which allow extending and personalizing an OpenFlow controller. However, most of them are only compatible with version 1.0 (OF 1.0), although the latest is version 1.1 (OF 1.1) with more interesting features [16]. NOX [17] was the first developed platform with OpenFlow support. Other platforms are Maestro [18], Beacon [19] or Trema [20], which are open-source, and ONIX [21], which is closed-source. The only available controller for OF 1.1, when we started our development, was a fork of NOX [22]. Therefore we use this version for our prototype, since we want to take advantage of the new features introduced in OF 1.1, such as, MPLS tagging. NOX provides an API to control OpenFlow switches from a controller.

1) *Forwarding rules*: There are different options that can be used with OpenFlow to control traffic from different users. Some of the alternatives are:

- **Native OpenFlow forwarding**: Create directly filters for the specific fields provided in the VN description. In this case, the selected matching fields must be unique for the entire network. This can be implemented directly modifying a native OpenFlow controller. A set of filters can be dynamically computed for the incoming flows.
- **Tunneling**: Create a tunnel between the ingress and egress points with one of the tunnel technologies supported by OpenFlow. The creation of the tunnel reduces the forwarding tables in the transit switches. Some of the alternatives are VLAN or MPLS tags.
- **Proxy**. Using Flowvisor [23] or similar tools, such as, ADVisor [24]. Different controllers have control over different network slices and the proxy isolates them and redirects the OpenFlow commands to the authorized controller. The slices can be defined from any field of the packet, for instance, network destination address.

We decided to use the native OpenFlow controller because it offers enough flexibility to differentiate the traffic of different clients. Tunneling introduces an extra mapping. Proxies introduce an extra delay, when processing packets, and introduce multiple controllers, which do not offer any added value to the customers since they cannot access them.

2) *Rules Insertion*: another consideration is how the paths are initialized or deleted. There are different ways:

- **Offline**: When the VN is initialized; all paths are computed and installed into the switches. This approach pre-establishes the flows, which removes any establishment delay. However, the switch has a limited capacity in the flow table and the pre-established flows will occupy table entries even if there is no traffic to forward.
- **On demand**: All required rules are installed after receiving the first packet of a flow. However, there is a flow establishment delay. The rules can be removed automatically with timeouts.

3) *Timeouts*: of installed path influences the size of the flow table and the scalability of the system. The options are: *permanent*, *fix timeout* and *idle timeout*.

- **Permanent**: a forwarding rule is always installed until the VN is removed. The rule needs to be deleted manually with an OpenFlow DELETE message.
- **Fix timeout**: a forwarding rule is removed after a constant time (hard timeout). This introduces undesired effects or losses if the flow is longer than the actual timeout.
- **Idle timeout**: a rule is kept in the switch table while there are packets that match the rule. The rule is removed after a certain time of flow inactivity. Idle timeout is not suitable for offline rule insertion because if the flow is not active, the rule might be removed before the flow starts.

Our first approach uses on demand rule insertion and idle timeout. The idle timeout is 60 seconds, which is the default ARP timeout in Linux systems.

B. Controller Architecture

We have implemented a prototype, which uses an OpenFlow controller to manage virtual networks. We use NOX [22] as a base platform, since it is widely used in the OpenFlow community and it supports OF 1.1. QoS constraints will be added in the next phase of the development. At this stage, LibNetVirt offers only traffic isolation.

The proposed architecture, for a L2 forwarding network, is shown in Fig. 2. The controller communicates with the driver over a TCP. The main components of the architecture are:

RulesDB is a database where the VN descriptions are stored. Every endpoint of the network, which has a VN associated to it, has one entry. This provides a fast lookup when a new packet arrives. Also, it provides a simple way to organize the VN information. In addition, we also keep a list of all the VNs. This list is used when removing the VNs and when checking the status of the controller.

Locator is a database where all the L2 addresses are stored. Currently it is kept in memory but it can be easily expanded to a distributed database or in a DHT, as suggested in [21]. The Locator learns the addresses from the incoming packets. If the destination is unknown, it sends an ARP request to all the endpoints of the VN. In this case, the ARP request is directly sent to the edge endpoints, in other words, the path

Fig. 2. VN NOX Controller architecture

that ARP request follows is source endpoint, controller and destination endpoint. This database is necessary for routing packets between two endpoints.

PathFinder is a database where the physical topology is stored. The topology information is obtained using the *Discovery* module in NOX, which sends Link Layer Discovery Protocol (LLDP) packets to discover the links between OpenFlow switches. LLDP packets contains the switch identity and using the incoming port information, it is easy to extract the network topology. PathFinder keeps a graph of the topology and updates the links when Discovery detects a new link. PathFinder computes the path between two endpoints. Currently, we use Dijkstra's algorithm [25] to find the shortest path, but other algorithms can be used.

Installed rules is a database where all installed OpenFlow rules in the switches are stored. There is no OpenFlow mechanism to ask what rules are installed; therefore it is necessary to keep track of the rules in order to remove them when it is necessary.

The packet processing procedure is as follows. A packet only arrives to the controller, with a *Packet In* event, when no rule is installed for a particular flow. The controller checks if the packet comes from an endpoint of an instantiated VN. If it is not, the packet is dropped. Otherwise, the correct VN is retrieved from the database (*rulesDB*), the path is computed with *PathFinder*, the rules are created and are stored in *Installed rules* and they are sent to the OpenFlow switches using the NOX API.

When removing a VN, the controller checks if there are any installed rules for it. Then, it sends a command to remove the installed rules on the switch and it removes the endpoint information from the *rulesDB* database. After checking all the endpoints, the VN is removed from the system.

IV. USE CASE: CLOUD NETWORKING

Nowadays, there is a tendency with cloud computing to externalize or concentrate the IT infrastructure from institutions into the cloud (public or private). A cloud [26] is a pool of

resources (computing, network and storage resources) which can be used and scaled up and down on demand without knowing the exact physical location. Some providers already offer *Infrastructure-as-a-Service* (IaaS), with main focus on computing or storage. Generally cloud resources are located in datacenters.

Cloud networking is focusing on the networking aspect of the cloud, which is in an earlier development stage compared to computing and storage. More dynamic functionalities and a common understanding of a network resource are needed. In this context, SAIL [27] has come up with a concept called *Flash Network Slices* (FNS). The goal of a FNS is to create Virtual Private Networks (VPN) on demand, providing Quality of Service (QoS) and dynamic reconfiguration properties.

A FNS can be viewed as a network resource which provides connectivity, isolation and quality of service for a user who wants to interconnect different resources inside or between datacenters or clouds. In order to simplify the view towards the user, a FNS is represented as a single node, which can act as a Layer 2 or Layer 3 VPN. Using this view, we can map the manipulation of a FNS with the single router abstraction used in LibNetVirt. This allows an easy manipulation of network cloud resources with a simple interface.

V. RELATED WORK

OpenStack Quantum [28] is a similar project that have started recently inside the OpenStack [29] ecosystem. The aim is to provide network connectivity as a service. It is also work in progress and their authors are focusing on OpenStack platforms to interconnect virtual machines. LibNetVirt has a larger vision, to be used also in other scenarios, such as, the interconnection between clouds and it permits to specify QoS constraints, such as, minimal bandwidth.

Keller and Rexford (in [12]) discuss about the convenience of the single router abstraction, allowing users to focus on their service, which is the desired goal in the cloud. We develop their concept in the VN view of the network.

Casado et al. (in [1]) propose an architecture for virtualizing the network forwarding plane. They separate the logical forwarding from the physical. We go a step further, abstracting the network topology to the user providing a generic interface to interconnect his resources and simplifying even more the view towards the user.

VICTOR [30] proposes an architecture for Virtual Machines mobility, which uses OpenFlow. It separates the logic control from the forwarding. All the edges nodes announce all the network prefixes. Our scope is slightly different, since their architecture offers mobility and we provide on demand provisioning and isolation.

NECs Programmable Flow [31] is a commercial solution to manage virtual networks in an OpenFlow-enabled network. It offers a wide and flexible set of features to manage such networks. LibNetVirt, even though offers less features, it enables managing legacy technologies, not only OpenFlow.

In the context of the Software-Defined Networking (SDN), several applications have been developed in the research

community. RouteFlow [32] converts an OpenFlow network into an IP routed network together with Quagga—an open source routing suite. Sharafat et al. (in [33]) implemented a MPLS-TE and MPLS VPN with OpenFlow. These approaches offer a path to move from the current close infrastructure to a Software-Defined Networking (SDN), but they do not simplify the network view.

Several attempts have been done in the last years to address the problem of lack of automation in the network operations, such as [2]–[5], which offer a complete platform for this purpose. Our approach consists of an open library that can be reused for several parties to achieve their goals. In addition, most of these platforms require a full graph of the desired network, without the level of abstraction required in the cloud.

VI. SUMMARY

LibNetVirt is an open source [13] library to simplify the network view towards the user independently of the underlying technologies. We believe that it provides the missing piece of network virtualization. It provides a simplified but powerful view towards the network. Its modularity permits to operate different technologies with the same API. We use the concept of a single node to represent the network. Only the definition of endpoints is necessary to create virtual networks.

In addition, we present a working prototype of the library using an OpenFlow-enabled network. This prototype allows creating networks on demand with a programmatic interface. This can be integrated with different cloud platforms, such as, OpenStack [29] or OpenNebula [34], or to used independently, to offer more services in cloud networking. We plan to do performance analysis as part of future work.

Finally, we analyzed some of the techniques used to create the necessary OpenFlow rules in the network switches to instantiate a virtual network with the single router abstraction.

ACKNOWLEDGMENTS

This work is being carried out under the scope of the European FP7 SAIL project under grant number 257448.

REFERENCES

- [1] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, “Virtualizing the network forwarding plane,” in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, ser. PRESTO ’10. New York, NY, USA: ACM, 2010.
- [2] “FEDERICA.” [Online]. Available: <http://www.fp7-federica.eu/>
- [3] “Autobahn: Geant2 bandwidth on demand (bod). geant2 deliverable dj3.4.2.” [Online]. Available: <http://www.geant2.net/upload/pdf/GN2-05-086v11.pdf>
- [4] X. Chen, Z. M. Mao, and J. Van der Merwe, “Pacman: a platform for automated and controlled network operations and configuration management,” in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT ’09. New York, USA: ACM, 2009, pp. 277–288.
- [5] T. Lehman, J. Sobieski, and B. Jabbari, “Dragon: a framework for service provisioning in heterogeneous grid networks,” *Communications Magazine, IEEE*, vol. 44, no. 3, pp. 84–90, march 2006.
- [6] “libvirt - the virtualization API.” [Online]. Available: <http://libvirt.org>
- [7] N. M. K. Chowdhury and R. Boutaba, “A survey of network virtualization,” *Comput. Netw.*, vol. 54, pp. 862–876, April 2010.
- [8] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, “Virtual network embedding with coordinated node and link mapping,” *Apr. 2009*, pp. 783–791.
- [9] T. Anderson, L. Peterson, S. Shenker, and J. Turner, “Overcoming the internet impasse through virtualization,” *Computer*, vol. 38, no. 4, pp. 34 – 41, april 2005.
- [10] G. Schaffrath et al., “Network virtualization architecture: proposal and initial prototype,” in *VISA ’09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. New York, USA: ACM, 2009, pp. 63–72.
- [11] X. Chen, Z. M. Mao, and J. Van Der Merwe, “ShadowNet: a platform for rapid and safe network evolution,” in *Proceedings of the 2009 conference on USENIX Annual technical conference*, ser. USENIX’09. Berkeley, CA, USA: USENIX Association, 2009, p. 3.
- [12] E. Keller and J. Rexford, “The “platform as a service” model for networking,” in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, ser. INM/WREN’10. Berkeley, CA, USA: USENIX Association, 2010, p. 4.
- [13] “Libnetvirt.” [Online]. Available: <https://github.com/danielstt/libnetvirt>
- [14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “OpenFlow: enabling innovation in campus networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.
- [15] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, “Network Configuration Protocol (NETCONF),” RFC 6241 (Proposed Standard), Internet Engineering Task Force, Jun. 2011.
- [16] “OpenFlow Switch Specification. Version 1.1.0 Implemented,” 2011. [Online]. Available: <http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf>
- [17] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, “NOX: towards an operating system for networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 105–110, Jul. 2008.
- [18] Z. Cai, A. L. Cox, and T. S. E. Ng, “Maestro: A system for scalable OpenFlow control,” *Rice University, Tech. Rep. TR10-08*, Dec. 2010.
- [19] “Beacon.” [Online]. Available: <http://www.beaconcontroller.net/>
- [20] “Trema. an open source modular framework for developing openflow controllers in ruby/c.” [Online]. Available: <https://github.com/trema>
- [21] T. Koponen et al., “Onix: a distributed control platform for large-scale production networks,” in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI’10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.
- [22] TrafficLab, Ericsson Research, “NOX Zaku with OpenFlow 1.1 support.” [Online]. Available: <https://github.com/TrafficLab/nox11oflib>
- [23] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, “Can the production network be the testbed?” in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI’10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–6.
- [24] E. Salvadori et al., “Demonstrating generalized virtual topologies in an openflow network,” in *Proceedings of the ACM SIGCOMM 2011*. New York, USA: ACM, 2011, pp. 458–459.
- [25] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.
- [26] National Institute of Science and Technology, “The NIST Definition of Cloud Computing,” Tech. Rep. [Online]. Available: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>
- [27] P. Murray et al., “D-D.1 Cloud Networking Architecture Description,” SAIL FP7 Project, Tech. Rep., 2011.
- [28] “OpenStack Quantum.” [Online]. Available: <http://wiki.openstack.org/Quantum>
- [29] “OpenStack.” [Online]. Available: <http://www.openstack.org/>
- [30] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song, “Enhancing dynamic cloud-based services using network virtualization,” *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 67–74, January 2010.
- [31] “NEC Programmable Flow.” [Online]. Available: <http://www.necam.com/pflow/>
- [32] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and M. F. Magalhães, “Virtual routers as a service: the routeflow approach leveraging software-defined networks,” in *Proceedings of the 6th International Conference on Future Internet Technologies*, ser. CFI ’11. New York, USA: ACM, 2011, pp. 34–37.
- [33] A. R. Sharafat, S. Das, G. Parulkar, and N. McKeown, “MPLS-TE and MPLS VPNS with openflow,” in *Proceedings of the ACM SIGCOMM 2011*. New York, USA: ACM, 2011, pp. 452–453.
- [34] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, “Virtual infrastructure management in private and hybrid clouds,” *IEEE Internet Computing*, vol. 13, pp. 14–22, 2009.