2012 European Workshop on Software Defined Networking

OpenFlow Virtualization Framework with Advanced Capabilities

Balázs Sonkoly, András Gulyás, Felicián Németh,
János Czentye, Krisztián Kurucz, Barnabás Novák, Gábor Vaszkun
Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
Email: {sonkoly,gulyas,nemethf}@tmit.bme.hu

***Abstract*—Due to its simplicity, transparency and performance, FlowVisor takes it all in today's virtualization tools for OpenFlow networks. We argue that this effectiveness comes at the price of intolerance towards diverse OpenFlow versions used simultaneously and also limited switch functionality. What is more FlowVisor based management frameworks cannot run and configure the network controllers. In this paper we present an integrated OpenFlow virtualization framework, that is capable of *(i)* running and managing multiple instances of OpenFlow switches with different forwarding capabilities and OpenFlow versions, *(ii)* running and configuring full controllers or network applications designed for controlling a virtual network under the management of the proposed framework, and *(iii)* configuring QoS in the network. Besides introducing the architecture we share implementation and deployment details in our prototype system.**

***Index Terms*—OpenFlow, virtualization, FlowVisor, QoS, configuration, network management**

I. Introduction

When the terms "OpenFlow" [1] and "virtualization" [2] appear in the vicinity of each other in some text, the word "FlowVisor" can also be anticipated with small chance of delusion. And indeed FlowVisor – used as a middle communication layer between the OpenFlow switches and the controllers– provides fast, easy and transparent decomposition of a given network into virtual slices and delegate slice-specific messages to the designated OpenFlow controller. The success of the FlowVisor concept is reflected by its widespread use in OpenFlow testbeds worldwide including for example GENI [3] and Ofelia [4].

The effectiveness of FlowVisor however comes at the price of inflexibility and limitations regarding the functionality of the switches in the network. These limitations are threefold. First, adding *functional extensions* to the switches – e.g. OAM functionality, unorthodox forwarding mechanisms, pseudowires or a new matching structure – is highly problematic, since both the switches and the OpenFlow protocol has to be updated. As FlowVisor is bounded to the OpenFlow protocol version, for every protocol update the FlowVisor has to be updated, as well. Secondly, in a network, there can be various OpenFlow switches and controllers *using different versions of the OpenFlow protocol*[1] at the same time. FlowVisor is not able to handle such situations, as it is inherently bound to a single version. Finally, the existing FlowVisor-based virtualization frameworks provide access only to the data plane and the FlowVisor(s). For example Expedient-type [5] of management tools, like the one used in Ofelia [4] the control plane and network controllers have to be managed by the users of the infrastructure. This approach highly obstructs the implementation of an integrated management framework.

In order to eliminate the drawbacks of the FlowVisor-based approach and other existing management frameworks, we have recently proposed a novel OpenFlow virtualization architecture and prepared a proof-of-concept prototype implementing its key elements [6]. Our proposed architecture gives a solution for running different versions of the protocol on the same network and same switches. We propose additional capabilities to run, stop and configure the appropriate controller as well, in order to further simplify the overall management of the network. Due to space limitations [6] contained only the high-level description of our architecture, here we share detailed information about the implementation and the prototype. Moreover, the architecture is extended by currently available QoS configuration capabilities in OpenFlow and we give a brief discussion on QoS functions required for future applications of the framework.

QoS provisioning has to be addressed in order to get a suitable architecture for carrier grade networks and service providers, therefore our QoS related contribution here have particular importance. QoS provisioning in packet switched networks has a very long history. Since the definition of the basic QoS requirements in the early 90's a plethora of studies considered QoS issues in packet networks and proposed scores of mechanisms and architectures to elaborate satisfactory solutions. In the light of volume of the QoS-related literature generated in the last two decades, one can be surprised that QoS provisioning is still an unsolved problem in today's packet networks. The reasons behind the phenomenon that QoS became such a chronic issue are definitely complex. However we

This work was performed in the High Speed Networks Laboratory at BME-TMIT. The authors would like to convey special thanks to András Kern, Dávid Jocha and Attila Takács from Ericsson Research.

[1] As time passes by this issue seems to sharpen according to the abundance of simultaneously used OpenFlow versions.

believe that the closed interfaces of the networking equipments significantly contributed to turning QoS provisioning into a permanent and classic problem. Since OpenFlow can support QoS, we believe that its open architecture coupled with an integrated management framework can activate and incentivize the research community to revisit QoS from a practical perspective and work out functioning systems, similarly as BitTorrent managed to implement multicast in the application layer.

The rest of the paper is organized as follows. In Section II the OpenFlow related network management approaches are summarized and a state of the art about the QoS support in OpenFlow is given. In Section III the proposed architecture is presented, followed by the details of our data models in Section IV. Finally our prototype system is introduced in Section V with numerous deployment details and Section VI concludes the paper.

II. Background

A. Network management in OpenFlow

Network management is an indispensable component of large-scale networks, however, OpenFlow has lacked a well-defined management framework. Recently, a new protocol has been proposed, namely OF-Config protocol [7], in order to take the first step toward a standard management framework. OF-Config is the configuration and management protocol for OpenFlow capable devices. Currently it is based on the Netconf [8], [9] network management protocol since it is more suitable for configuration management than traditional SNMP. The proposal also gives data models of OpenFlow components using the Yang [10] data definition language.

In spite of lacking standard management tools in OpenFlow, several OpenFlow capable testbeds were established in the recent years in the US and Europe, as well. These facilities provide specific control and management interfaces to experimenters focusing on the configuration of data plane and FlowVisor [2]. For example, Expedient [11] is the configuration tool used currently in Ofelia, and it was used by GENI previously. Now the network configuration in GENI is based on FOAM [12], [13]. Another important feature of OpenFlow is the capability for provisioning network virtualization. Currently, FlowVisor is the widely used component providing light-weight network virtualization, or network slicing. Naturally, the management tools have to be capable of configuring and managing the operation of this component, as well. A widely used companion tool of Expedient is Opt-In Manager [14] managing information on flowspaces owned by users and on running experiments. Slicing of network resources is solicited by pushing down the flowspace information to FlowVisor.

B. Quality of Service in OpenFlow

The OpenFlow community realized the importance of Quality of Service support by enriching the 1.0 version of protocol with some QoS capabilities [15]. However, even the latest, 1.3 version has only a limited QoS support through a simple queuing mechanism [16]. In version 1.0, packets can be forwarded to queues of output ports by the optional enqueue action, which was transformed into the optional set-queue action in version 1.2. Controllers can query various queue statistic and queue configuration parameters through the OpenFlow protocol, but the protocol has no support to create queues or alter the behavior of existing queues. Therefore queue configuration needs to take place outside of the OpenFlow realm.

Although OpenFlow protocol does not support creating or modifying queues, an OpenFlow capable switch can be queried to report the queues attached to a specific port, and the guaranteed minimum rate associated with a queue. Additionally, after version 1.2, it can report to controllers the maximum rate as well. The minimum and the maximum rates are, of course, not enough to build a moderately complicated QoS scenario, but traffic queues can have other characteristics that cannot be queried. For example, an OpenFlow controller is unable to distinguish a small sized RED queue with zero minimum threshold from an enormous FIFO queue causing bufferbloat.

Instead of adding switch configuration instructions to the OpenFlow protocol, the Open Networking Foundation created an auxiliary protocol called OF-Config [7]. Its first version supports configuring an OpenFlow v1.2 capable switch.[2] From QoS point of view, it can be used only to remotely set minimum guaranteed and maximum rates of queues. Interestingly, OF-Config should be used to configure queues, but OpenFlow should be used to configure meter tables.

III. Architecture

To handle all the missing pieces listed in Section I, we have defined an integrated OpenFlow virtualization framework. The architecture is shown in Fig. 1. It is based on flexible virtualization capabilities of the data plane elements. Such elements are able to run multiple instances and versions of OpenFlow switches and to provide dedicated contexts for these switch instances. They also comprise of a switch logic that implements the mapping between the physical interfaces of the switches and the virtual ones belonging to the different contexts. This lower layer, the virtualization entity of the data plane elements could belong to the Infrastructure Provider and it is capable of mapping virtual networks into the physical one. The Infrastructure User, e.g., a service provider, gets a virtual network which is a configurable view of the physical network. Similarly to the IEEE 802.1Q Virtual LANs, our virtualization layer supports two types of mapping: exclusive mapping implements a one-to-one relation between a physical and a virtual interface, while it also allows mapping multiple virtual ports to a common physical trunk one. In this latter case, to all packets an additional link specific identifier for the virtual network, for instance a VLAN tag or an MPLS label, must be added to make distinction between traffic belonging to the different virtual instances.

[2]The latest version supports OpenFlow v1.3.

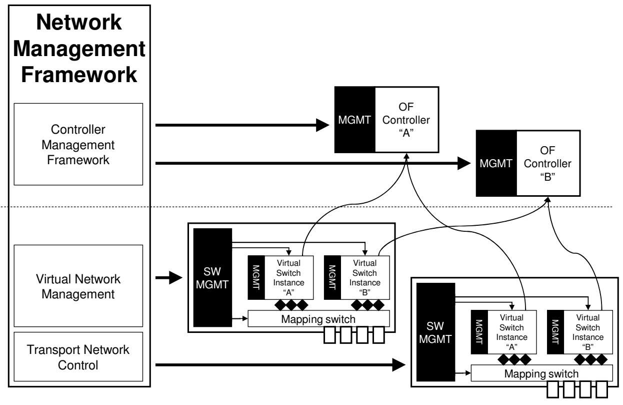
Network Management Framework
Controller Management Framework
Virtual Network Management
Transport Network Control
MGMT
OF Controller "A"
OF Controller "B"
SW MGMT
Virtual Switch Instance "A"
Virtual Switch Instance "B"
Mapping switch

Fig. 1. Integrated OpenFlow virtualization architecture.

The proposed integrated architecture implements the essential management and control functions for managing these enhanced virtualization capable switches. On the one hand, the network management system configures the switches: instantiates virtual switch instances, starts/suspends/stops them, assigns the physical ports to the virtual ones and optionally may download new versions of the OpenFlow virtual switches. On the other hand, it also maintains the physical topology of the managed network. For instance, it may reconfigure the physical-to-virtual port assignment as a response of a physical failure. Important to note that such reconfiguration happens without any effects on internal configurations of the virtual switches.

This framework also provides methods on running full controllers or network applications designed for controlling a virtual network under the management of the proposed framework. Such methods comprises of (1) loading and starting previously downloaded third party controllers, (2) configuring the previously instantiated virtual switches to connect to the just started controller, (3) monitoring different characteristics of the running controller and last but not least (4) maintain/stop/restart the controller as needed. A natural extension is where not only virtual node operation system is given for running a controller, but the whole network operating system is implemented by the framework. This allows implementing only the network application instead of a full controller; which is in turn enabler of the Software Defined Networking.

Similarly to the controller provisioning framework, lightweight network virtualization methods, like FlowVisors can also be managed with this framework.

QoS provisioning is a multi-stage process in a virtualized networking environment. The Infrastructure User first needs to have resources with QoS guaranties to be able to share them further to its users. There are two extremities from architectural point of view. In the centralized case, the Infrastructure User relays every QoS request to the Infrastructure Provider and dedicates the resource received to the end user. In the distributed case, the Infrastructure Users are initialized with long-term QoS guaranteed resources and it is their responsibility to handle locally QoS requests of the end users and renegotiate guaranteed resources with the Infrastructure Provider. The centralized case can easily adapt the results of rich literature of hierarchical packet schedulers (e.g., [17], [18]), whereas an already existing OpenFlow system can be more easily ported to the virtualized environment in the distributed case.

IV. Data models

The management interfaces of our framework is based on the Netconf [8], [9] protocol and we use Yang [10] as data modeling language. This section is devoted to give some details on the data models that we have designed for the relevant network elements.

A. Yang model for OpenFlow switches

The structure of the data model can be broken down to three main parts: configuration and current state data, information from the OF switch instance (e.g. descriptions, like manufacturer, or flow table statistics), remote procedure calls (e.g. start/stop instances, create virtual interfaces). Listing 1 shows the basic schematic of the data model while Listing 2 presents the RPCs.

The top-level container handles the various OpenFlow reference switch instances. Each instance is distinguishable through their IDs and it has two containers. One holds crucial configuration parameters that are required for successful execution, while the other is only significant if the switch is capable of communicating with the Netconf daemon.

The *config container* holds the basic parameters of the switch instance, such as path of the binaries (ofdatapath, ofprotocol, dpctl), the parameters of the corresponding controller and interface information (eth-if leaf-list). The data model is capable of handling virtual Ethernet interfaces, as well. In addition, the container holds information on the state of the switch instance (e.g., PID values, status).

The *datapath container*, is capable of gatehring statistical data by querying the OpenFlow switch directly. This functionality requires switch extensions that support the communication with the Netconf server (e.g., through a Unix Domain Socket). The query-able parameters include several descriptions and statistics of the flow tables. The statistics are held in a list, which is intended to have dynamic size so that only active tables could be queried.

The data model has four basic RPCs, namely, start, stop, dpctl, and create-veth as it is shown in Listing 2. The first two handles the starting and stopping of the OpenFlow instances, while dpctl is an all-purpose manual controller RPC which directly calls the dpctl on the given dpctl-path. And finally, the create-veth RPC creates pairs of virtual interfaces that can be used for virtualization of the switch instances.

Managing QoS queues is achieved by the use of three RPCs, more specifically, get, set and delete. A queue is identified by the combination of an ID and a port number. The only available QoS parameter is the bandwidth limit. *Config true list* is not suitable for handling queue parameters because it stores the values in the Netconf daemon and Netconf server

Listing 1. Basic schematic of the Yang model.

```
container of−sys{
  list of−instance {
    leaf id {
      description "ID of the current OF inst."; }
    container config {
      description "Container for config data";
      // basic parameters
      leaf datapathid { }
      leaf version { }
      leaf status{ }
      leaf dp−pid { }
      leaf of−pid { }
      leaf ofdatapath−path { }
      leaf ofprotocol−path { }
      leaf dpctl−path { }
      leaf−list eth−if { }
      leaf dp−of−socket { }
      ...
      // controller parameters
      leaf controller−type{ }
      leaf controller−address{ }
      leaf controller−port { }
    }
    container datapath {
      description "Data from datapath";
      // OF switch parameters
      leaf mfr_desc{ }
      leaf hw_desc{ }
      leaf sw_desc{ }
      leaf dp_desc{ }
      leaf serial_num{ }
      leaf id{ }
      ...
      container pipeline{
        list flow_table{
          // flow table parameters, statistics
          leaf table_id{ }
          leaf name { }
          leaf wildcards{ }
          leaf match{ }
          leaf instructions{ }
          leaf write_actions{ }
          leaf apply_actions{ }
          leaf config{ }
          leaf max_entries{ }
          leaf active_count{ }
          leaf lookup_count{ }
          leaf matched_count{ }
        }
        container table−sum{
          leaf active_sum{ }
          leaf lookup_sum{ }
          leaf matched_sum{ }
        }
      }
    }
  }
}
```

cannot detect queue reconfiguration. However, *config false list* is not suitable either because of a bug in Yuma toolkit[3]. As workaround we create a family of RPCs. The get-queue RPC

[3]Yuma toolkit cannot handle config false parameterized lists, i.e., the list which holds the statistical data of the flow tables, with altering number of elements. A list element is created when create RPC is called on the list, but a config false list is not create-able. No function is generated for a config false list which could define the required number of elements and then creates the elements in a for/while cycle.

Listing 2. RPCs of the data model.

```
/* basic RPCs */
rpc of−start { input { leaf instance { } } }
rpc of−stop  { input { leaf instance { } } }
rpc dpctl    { input { leaf instance { }
                       leaf param { }    } }
rpc create−veth {
               input { leaf pairs { }    } }

/* QoS−related RPCs */
rpc set−queue { input { leaf q_id { }
                        leaf bandwidth { }
                        leaf port { }    } }
rpc get−queue { output{ list queues {
                          leaf port { }
                          leaf q_id { }
                          leaf bandwidth { } }
                                         } }
rpc delete−queue {
               input { leaf port{ }
                       leaf q_id{ }      } }
```

lists all the configured queues, which are queried directly from the switch. The RPC itself first determines the number of queues and then it creates the required number of elements and queries their values from the switch.

B. Yang model for NOX controllers

The Yang data model for NOX controllers is presented in Listing 3. It shares similarities with the data model of OpenFlow reference switch, such as handling multiple instances, starting, and stopping each instance.

Listing 3. Basic data model of NOX controller.

```
container nox−sys {
  list nox−instance {
    leaf id { }
    leaf version { }
    leaf path { }
    leaf config−file−path { }
    leaf noxapp−path { }
    leaf connection−type { }
    leaf port { }
    leaf−list noxapp{ }
    leaf status{ }
    leaf pid { }
  }
}
rpc nox−start { }
rpc nox−stop { }
```

For appropriate functionality of a NOX controller the following parameters are essential: path, connection type (e.g., tcp, ssl), listening port, and at least one NOX application. Due to compatibility between NOX versions and OpenFlow protocol versions, the path has been divided into three different path variables (binaries path, configuration file path, and NOX applications' path). Instead of a simple *leaf*, the model defines a *leaf-list* for NOX applications in order to run them simultaneously. A status and PID leaf were also added for monitoring.

C. Comparison with OF-Config

There are several similarities between the data models of our management framework and the official OF-Config protocol. Both solution support virtualization of logical switches, and offer query of statistical data[4]. The main difference is that the OF-Config 1.1 supports a wider range of query-able statistical data, and also supports setting and querying of configuration data (e.g. flow entries). Another difference is the way the virtualization is implemented. Although both solutions allow creation of logical switches, our implementation is more flexible in handling different version of OpenFlow switches. The OF-Config defines every resource in a uniform way and then distributes it between the logical switches. This causes version dependency between the switches, because each switch handles their resources a little different. Our solution is completely version independent and the only resources shared between the different switches are the physical and virtual interfaces. The framework allows usage of not just different versions of OpenFlow protocol, but also customized switch codes.

V. Prototype Implementation

We have prepared a proof of concept prototype making use of open source components focusing on key elements of the presented architecture. The source codes are released as open source and can be publicly available [19]. We chose OpenNMS [20] as the management framework because it is open source, highly flexible and can easily be extended in a modular way. Two novel components were added, namely, the virtual network management module and the controller management module.

On the one hand, virtual network management component is responsible for configuring virtualization layer mapping switches via the Netconf protocol (with Yang as data modeling language) to establish virtual topologies. Currently, the distinction between packets of different virtual instances is based on VLAN tags managed by corresponding flow entries[5]. In addition, virtual switches with different forwarding capabilities can be instantiated at Infrastructure User's layer and configured with given parameters (e.g. controller address), such as OpenFlow v1.0 and v1.1 reference switches, Open vSwitch [21], OpenFlow switch with greedy forwarding [22], and Open vSwitch with Bloom filter based forwarding. This component is also responsible for creating virtual Ethernet pairs and connecting the virtual switches with the mapping switch.

On the other hand, basic configuration management capabilities were implemented in the controller management module for (re)starting and stopping NOX controller instances as demonstration. It is worth noting that the framework is not limited to NOX but other controllers (such as Beacon, Maestro, Trema, etc.) can also be managed and configured by this way.

[4]OF-Config 1.0 lacks this function; it was only added to version 1.1.

[5]In spite of using VLAN tags, we do not get a fix slicing scheme because the network management component can dynamically reconfigure it.

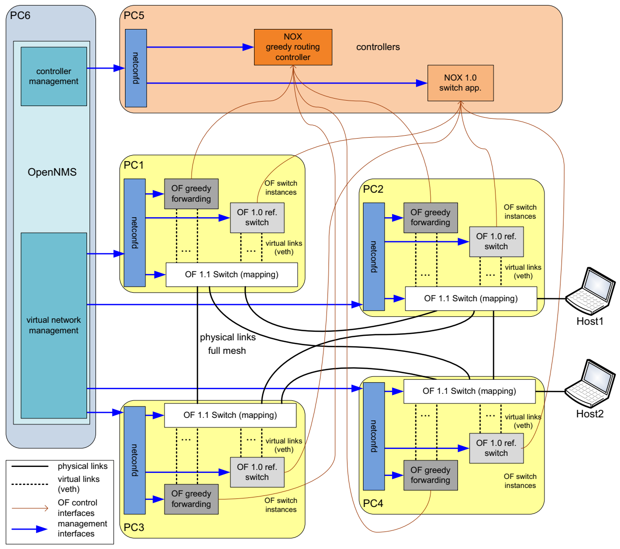

Fig. 2. Small testbed showing connections among physical devices and software components running on them.

By this approach, Infrastructure Users can upload and run their own controller applications or services under the control of the management framework. Moreover, Infrastructure Provider can give common controller applications that can be used by different Infrastructure Users.

Netconf management interfaces of our prototype are based on the open source Yuma tools [23] providing Netconf server (netconfd) and client (yangcli), respectively. We integrated client components with OpenNMS and added agents to switches and controllers. The prototype system currently operates one Netconf agent per PCs and the logical network components running on a PC are configured via a common agent.

We verified the capabilities of the framework with our prototype by running experiments on the topology shown in Fig. 2. Data plane consists of four PCs (PC1-4) running a mapper switch, a switch with greedy forwarding, and an OpenFlow 1.0 reference switch, respectively. These PCs are connected physically in a full mesh. The NOX controller of the greedy domain and a standard MAC learning NOX application are running on PC5, while OpenNMS is hosted on PC6. We use a separate, out-of-band control and management network.

The testbed system operates with single Netconf agent per PCs, however, the extension with respective agents or NMS instances for virtual domains is straightforward. Additionally, we have ported our prototype into OpenWRT firmware and instead of PCs running switch instances physical devices can also be used. It is worth noting that we use software switches on TP-Link TL-WR1043ND devices (or PCs) running in user space and these hardware devices have low-performance CPUs. Of course, running multiple switch instances results in performance degradation, however, we have found that even these low performance devices are capable of operating without significant degradation[6]. As we are working on a proof-of-concept prototype, we do not focus on performance at this stage. Our experiments are conducted in order to validate the operation of the network and the components.

This small testbed is capable of creating and managing multiple virtual networks consisting of different types of OpenFlow switches. More specifically, we have conducted several experiments with OF v1.0 and v1.1 switches and our custom switches implementing greedy forwarding mechanism. The network can easily be configured from our OpenNMS-based management system. In our scenarios, we have sent different video streams through the same physical network but using different virtual topologies. The experiments validated the behavior of the framework in all cases.

As an illustration for the configuration mechanism, creating a queue with QoS guarantees generates the following chain of events. First, the network operator sets the parameters of a queue with minimal guaranteed rate in the QoS module of OpenNMS. Then the OpenNMS sends a set-queue Netconf message to the target node's Netconf server. The Netconf server transforms thise message into a *dpctl* command. Having received this dpctl command, the OpenFlow switch further maps the configuration request to a *traffic control* (tc) command that finally creates the proper queue in the kernel. (Although the Netconf server could directly issue tc commands, but then the software switch should need to be notified about the new/changed queue.)

VI. Conclusion and Future Work

In this paper we introduced an integrated virtualization and management framework for OpenFlow networks, which can accommodate OpenFlow version diversity, can configure OpenFlow controllers besides datapaths and provides support for QoS. Our experience with the implementation and deployment of the system was also shared. By this time the prototype system is capable of performing basic managements tasks, there are numerous possibilities for future improvement.

One future direction is to extend the framework with additional QoS capabilities. Although, it seems the Open Networking Foundation continuously enhances QoS support in OpenFlow protocol and its auxiliary OF-Config protocol with each new version. However, until now, it has focused only on provisioning traffic speed guaranties. Providing low-latency connections to, e.g., VoIP traffic with priority queuing is not currently possible. Similarly, the protocols do not allow to configure or query active queue management properties, for example, to avoid massive packet loss due to TCP synchronization. If we incorporate the support to these QoS techniques, then Infrastructure Users and their OpenFlow applications will benefit from the new QoS guaranties even without direct protocol support. In general, a QoS architecture for OpenFlow with sophisticated QoS functions is a challenging research direction. Our plans on future work in this area are summarized in a companion paper [24].

We also address the extension of our management system with topology visualization capabilities. This new module will include the visualization of the physical and virtual topologies, link states and traffic information, respectively.

[6]More complex network devices currently support running different instances or contexts connecting them by virtual links. On these hardware platforms, acceptable performance can also be achieved.

References

[1] Open Networking Foundation, "Software-defined networking: The new norm for networks," ONF White Paper, Apr. 2012.
[2] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, N. McKeown, and G. Parulkar, "FlowVisor: A network virtualization layer," OpenFlow Switch Consortium, Tech. Rep. OPENFLOW-TR-2009-1, 2009. [Online]. Available: http://www.openflow.org/wk/index.php/FlowVisor
[3] "Exploring networks of the future." [Online]. Available: http://www.geni.net/
[4] "Ofelia, OpenFlow in Europe." [Online]. Available: http://www.fp7-ofelia.eu
[5] "Cross-aggregate Resource Reservation with the Federated GENI API." [Online]. Available: http://groups.geni.net/geni/attachment/wiki/GEC8DemoSummary/GEC8-Stanford-Integration.pdf?format=raw
[6] B. Sonkoly, A. Gulyás, J. Czentye, K. Kurucz, G. Vaszkun, A. Kern, D. Jocha, and A. Takács, "Integrated OpenFlow virtualization framework with flexible data, control and management functions," in *Proceedings of IEEE INFOCOM 2012 (Demo)*, Orlando, Florida, USA, 2012.
[7] Open Networking Foundation, "OpenFlow configuration and management protocol 1.0 (OF-Config 1.0)." [Online]. Available: https://www.opennetworking.org/standards/of-config-10
[8] R. Enns (Ed), "NETCONF configuration protocol," Internet Engineering Task Force, RFC 4741, Dec. 2006.
[9] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman (Eds), "Network configuration protocol (NETCONF)," Internet Engineering Task Force, RFC 6241, Jun. 2011.
[10] M. Bjorklund (Ed), "YANG - a data modeling language for the network configuration protocol (NETCONF)," Internet Engineering Task Force, RFC 6020, Oct. 2010.
[11] J. Naous, "Expedient: A pluggable platform for geni control framework." [Online]. Available: http://yuba.stanford.edu/~jnaous/expedient/
[12] GENI project contributors, "FOAM." [Online]. Available: http://groups.geni.net/geni/wiki/OpenFlow/FOAM
[13] J. Smift, "GENI, openflow update – FOAM," presentation slides, Nov. 2011, Kansas City, Missouri.
[14] OpenFlow Consortium contributors, "Opt-in manager." [Online]. Available: http://www.openflow.org/wk/index.php/OptIn_Manager
[15] OpenFlow Consortium, "OpenFlow switch specification, version 1.0.0," Dec. 2009. [Online]. Available: http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf
[16] Open Networking Foundation, "OpenFlow switch specification, version 1.3," Apr. 2012. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf
[17] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 4, pp. 365 –386, aug 1995.
[18] J. Bennett and H. Zhang, "Hierarchical packet fair queueing algorithms," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 675–689, 1997.
[19] "Integrated OpenFlow virtualization framework." [Online]. Available: http://sb.tmit.bme.hu/mediawiki/index.php/OpenFlowVirtualizationFramework
[20] "OpenNMS." [Online]. Available: http://www.opennms.org/
[21] "Open vSwitch." [Online]. Available: http://openvswitch.org/
[22] B. Sonkoly and A. Gulyás, "Overhead-free routing with OpenFlow," in *Ofelia Workshop, ECOC*, Geneva, Switzerland, Sep. 2011.
[23] "Yuma: a YANG-based unified modular automation toolkit." [Online]. Available: http://www.Netconfcentral.org/yuma
[24] B. Sonkoly, A. Gulyás, F. Németh, J. Czentye, K. Kurucz, B. Novák, and G. Vaszkun, "On QoS support to Ofelia and OpenFlow," in *Proceedings of European Workshop on Software Defined Networks (EWSDN)*, Darmstadt, Germany, 2012.