High-Performance Virtualized SDN Switches for Experimental Network Testbeds

Richard Cziva[a,*], Jerry Sobieski[b], Yatish Kumar[c]

[a] *University of Glasgow / NORDUnet*
[b] *NORDUnet / GEANT*
[c] *Corsa Technologies*

Abstract

Software-Defined Networking (SDN) has been in the spotlight of the networking research community by promising programmability and centralized control over the entire network infrastructure. During the recent years, SDN's most wide-known realisation, the OpenFlow [1] protocol, has gained support from many software and hardware switch vendors resulting in SDN deployments at campus networks and industry systems. To support experimentation and research projects, experimental testbeds have also started introducing SDN resources to support research and application development in this new service model.

However, despite their widespread adoption, current SDN resources offered by experimental testbeds do not support virtualization at the device level, resulting in inefficient provisioning, as well as functional and performance limitations. To solve the aforementioned problems, GEANT and Corsa have been working together to deliver fully virtualized OpenFlow resources as part of the GEANT Testbed Service, one of Europe's large-footprint experimental network testbed facilities. The proposed virtual switches can be provisioned using a human-readable domain specific language and provide high-performance (up to 100Gbit/s) packet forwarding with accurate resource isolation between multiple OpenFlow-compliant virtual switches.

Keywords: Software Defined Networking, OpenFlow, Corsa, Virtual Switch Instances, Domain Specific Language, Experimental Network Testbeds

1. Introduction

Software-Defined Networking (SDN) is a promising approach to centralize the control plane of a network and decouple the control place from a data plane that only performs basic packet processing and forwarding. The OpenFlow [1] protocol, as SDN's

*I am corresponding author

Email addresses: r.cziva.1@research.gla.ac.uk (Richard Cziva), jerry@nordu.net (Jerry Sobieski), yatish@corsa.com (Yatish Kumar)

first and most popular realization, has been used in countless research and commercial systems as an open protocol to communicate between between SDN controllers (e.g., Ryu, ONOS, Open Daylight, Quagga, Faucet, etc.) and the data plane of the network devices. Following the widespread adoption and the trending research around SDN, OpenFlow resources have been introduced to experimental network testbeds. OpenFlow switches can now be provisioned in testbeds such as the GENI, the GEANT Testbeds Service, and numerous FIRE projects.

While in today's experimental testbeds compute resources and transport resources have been offered as virtualized resources such as Virtual Machines and VLANs or other virtual circuits, this has not been the case for SDN/OpenFlow resources. First, traditional OpenFlow switches lack virtualization support, resulting in OpenFlow switches being treated as single physical boxes that work with a single controller. This limits the sharing of the physical device, and often pushes excess complexity into the controller and user applications in an effort to support secure sharing of the hardware. Some vendors support multiple logical OpenFlow protocol instances that allow ports to be mapped to each instance and allow a separate controller to be linked to each instance. While this is an improvement, these partitioned sharing models still require flows to match against physical port IDs - which violates virtualization by exposing physical port information and tie the application (the controller in this case) to specific physical port assignments. This means that migrating a logical switch from one physical switch to an other can invalidate many flow entries that rely on physical port numbers (it is the case for many L2 switching applications). Also, the physical port delegation to specific OpenFlow instances does not allow port sharing - a big problem if there are high capacity backbone links (10/40/100Gbps) terminated on the switch.

Moreover, as most OpenFlow switches have not been designed for virtualization, they lack flow space and performance isolation. While a viable solution is to virtualize traditional switches using a transparent software layer (e.g., Flowvisor [2]) on top of the physical devices, these solutions introduce substantial latency and complexity to flow management since each OpenFlow interaction must transit the proxy controller, and inserting and deletion of each flowspec must be inspected and authorized. These software approaches further do not support fine-grained resource allocation between virtual switches. We believe and advocate that virtualization should be supported by the physical devices, the same way as CPUs today support virtualization as a hardware feature (e.g., VT-X feature in Intel CPUs). Moreover, we believe that there should be a common, platform-independent description language that can be used to describe and provision OpenFlow resources in experimental research testbeds.

In this paper, we will introduce Virtual Switch Instances (VSI)s. VSIs are the result of collaboration between the academic network research community (GEANT network consortium, and the Nordic Universities Network NORDUnet), and an SDN switch vendor (Corsa Technologies). The contribution of this work is twofold:

1. First, we present the Virtual Switch Instances and a hardware appliance designed with full OpenFlow switch virtualization in the focus.
2. Also, we present a platform-independent description language that is used to define virtual OpenFlow switch resources. We evaluate the proposed system in a European experimental network testbed (GEANT's GTS project).

The remained of this paper is organized as follows. In Section 2 we introduce the Virtual Switch Instances with their benefits for users and providers. In Sections 3 and 4 we introduce the advances we made to implement VSIs in hardware and software, respectively. In Section 5 we provide evaluation results using high traffic load and multiple virtual switches created on a single physical device. In Section 6 we compare various experimental testbeds by their SDN/OpenFlow resource offerings. In Section 7 we conclude the paper.

2. Virtual Switch Instances

In this Section, we are introducing virtualized OpenFlow switches, called Virtual Switch Instances (VSI)s. VSIs, if implemented rigorously, allow hardware switching infrastructure to be shared by multiple independent SDN applications (controllers) - each perceiving their own SDN fabric. The two key contributions of VSIs are the following:

- Each VSI provides its own OpenFlow protocol model context, such as a controller protocol stack, flow space with separate flow entries, meters, flow groups and counters.
- The switch virtualizes the port mapping in the fast path. This port mapping allows each VSI to be virtualized and separated from physical port specifications.

The following two subsections detail how traffic is mapped to VSIs and how providers and users benefit from VSIs.

2.1. Mapping traffic to VSIs

Mapping traffic to VSIs is one of the key aspects of the proposed system. In our case, each packet is inspected upon arrival and the port/outer VLAN tag is swapped to a VSI instance identifier and a virtual port identifier. From this point, the user's flow specifications match against the virtual port identifier and not the physical one where the packet entered the device. As shown for example in Figure 1 at physical port 5, all packets arriving with VLAN id 10 will be directed without the VLAN tag to VSI 2 on virtual port 1. While we use outer VLAN tags (applied by an edge router connecting hypervisors and external connections to the switch) to allow multiple users sharing the same physical links, one could assign untagged physical ports directly to VSIs, as

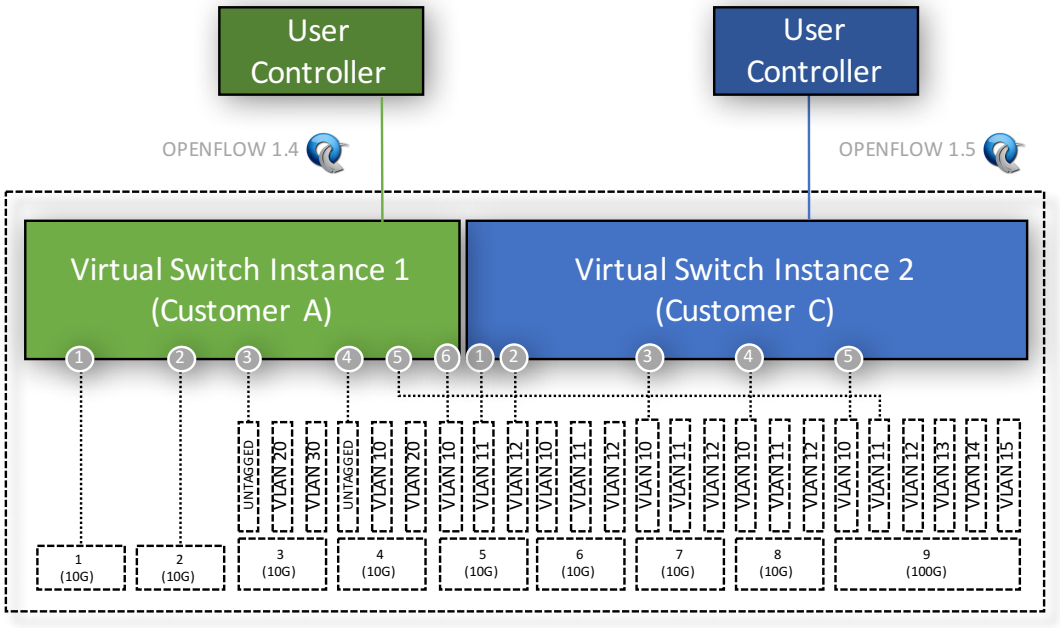

Figure 1: Mapping traffic to different VSI contexts.

shown at port 1 and 2 in Figure 1, packets can also arrive untagged and mapped to VSIs directly without VLAN tag removal.

Mapping traffic to VSIs is supported in high speed, equivalent to an MPLS swap function, where frames can be remapped at line rate, exceeding 100Gbps. When packets leave the device (outbound frames), the port mapping is reversed, where the VSI identifier and the output port are remapped to physical port and an outer tag. The push or pop of the outer tag is configurable and can be disabled - in this case port sharing is still supported, but VSIs will see the native frame information.

2.2. Benefits

2.2.1. Benefits for Providers

The key benefit for providers is that VSI are "well bounded" service objects. Providers can allocate VSIs to arbitrary users and can securely constrain the range of traffic those VSIs can see or manipulate and can efficiently manage the switching capacity and performance allocated to each VSI. Just as Virtual Machines present an effective means for cloud service providers to allocate computational resource securely, so Virtual Switch Instances allow network service providers to allocate switching capability inside their network securely. The user sees their own customized dedicated switch fabric and a full network flowspace, the provider sees a bounded manageable and secure service object they can offer to arbitrary users.

Further, as VSIs are fully virtualized resources that, by merit of virtual port mapping, have flowspecs that match against virtual ports - not physical ports. Thus relocating the VSI to reflect network events or management requirements can be performed by copying the VSI context and simply remapping the virtual ports. The VSI virtual ports do not change, and thus the user controller/application does not need to rediscover its topology and rewrite all of its flow rules. The user application may see a brief interruption (on the order of a few milliseconds such as for a protection switch) but this capability allows providers to migrate and groom VSIs across physical platforms without changing the logical topology of the user network or otherwise breaking the SDN application. This supports maintenance windows, efficient infrastructure management, and fault tolerance.

The virtual port mapping to VSIs also enables port sharing, as a single backbone port can carry virtual transport links to/from multiple VSIs, i.e. entire physical ports are not delegated to VSIs, but rather the flows or virtual circuits (VLAN, MPLS) can be multiplexed across a single physical backbone link and mapped to their respective VSIs and virtual ports.

2.2.2. Benefits for Users

Each VSI is perceived by the user as a complete and dedicated switch fabric. Each VSI has its own control link and protocol stack, thus allowing the user to employ any controller software they wish independent of any other VSIs. The configuration of the Virtual Switch fabric instance itself can be defined by the user (such as number of ports, version of the OpenFlow protocol used at the VSI, flowspec memory requirements, etc.) The GEANT Testbeds Service (GTS) uses a Groovy based domain specific language to describe these virtual switch objects and their attributes. This switch description is then sent to GTS via the GTS API Reserve() primitive and a VSI is allocated in the desired location with the specified attributes. Apart from great functional convenience of this service model and the VSI capabilities, VSIs operate at line-rate (up to 100Gbit/s) - thus allowing true high performance SDN applications to be easily provisioned across multiple provider domains globally.

The VSI allows user networks (SDN application domains) to span multiple provider infrastructure networks without complex intervening provider control proxies that would otherwise be needed to inspect, authorize, and arbitrate every flow rule before installing it. Users can request VSIs in any infrastructure domain, and terminate virtual circuits or other inter-domain transport links on those virtual ports. Providers need only authorize and provision the initial VSI configuration, then the user has direct control of that VSI.

An interesting side benefit of virtual port mapping means that VSIs can be sized as necessary to an application's requirements - small pipelined virtual network functions such as firewall filters or traffic shapers can make use of VSIs with only a few virtual ports, or a larger network aggregation VSI may be defined to have 100 [virtual] ports.

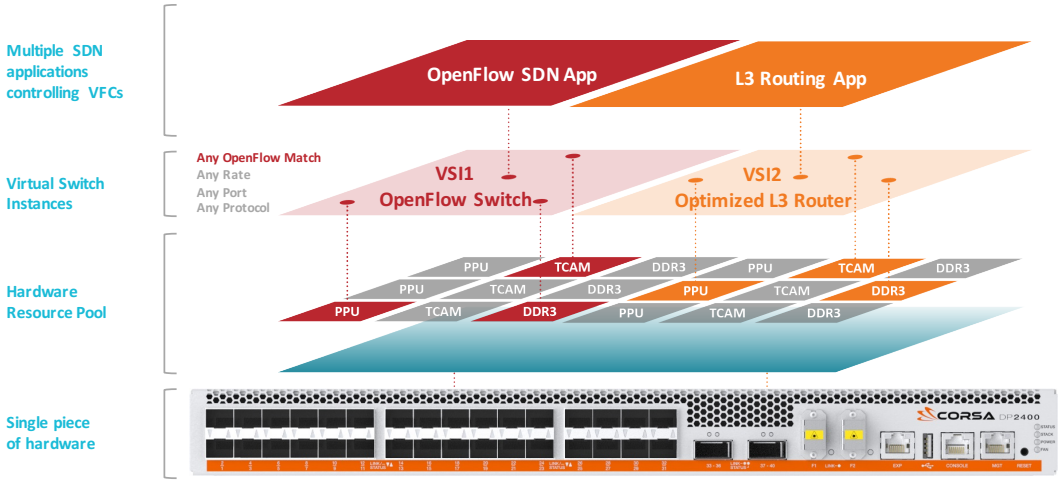

Figure 2: Corsa DP2400 high level architecture presenting two VSIs

The virtual port mapping allows the VSIs to establish as many ports as they require - possibly even exceeding the number of physical ports on the switch. The total aggregate capacity of the switch is the theoretical limiting factor on port and switching allocations.

3. Hardware Design

The proposed system relies on Corsa 2000 series SDN switches with a pre-production firmware that has been developed for this project. The firmware will be available for public later this year. The DP2000 devices provide 100G non-blocking throughput with a capacity for over a million of OpenFlow flow entries. The devices support the latest versions of OpenFlow (1.5) and have a massive packet buffers. Moreover, a state-of-the-art traffic metering, policing and shaping have been introduced in order to support multiple virtual switch instances with different bandwidth requirements.

A high-level architecture of the DP2400 device can be seen in Figure 2. As shown, the device provides a pool of hardware resources (also called as "underlay virtualization") for VSIs - such as Packet Processing Units (PPU)s, Ternary Content-Addressable Memory (TCAM) units and DDR3 memory. Depending on the configuration of the device, VSIs can allocate resources from this pool.

3.1. Hardware challenges with Virtualization

Virtualization can pose significant challenges on hardware. In this Section, we present the top three challenges that have been solved by Corsa: the flexibility of OpenFlow tables and advances in virtual QoS, metering and statistics.

3.1.1. Number of OpenFlow tables

A non-virtualized OpenFlow switch tends to define an upper limit on the number of OpenFlow tables it can support based on a number of factors, including total state storage in TCAM and other look up memories for all the active tables, as well as total internal program memory and register storage allocated to hold the match action structure for each of the provisioned tables [3]. Typical numbers for such limits range from 8 tables to 32 tables per hardware switch. In order to virtualize and multiply the number of active pipelines both the program memory space, as well as the active table state storage need to be scaled by a factor of NxT where N is the number of virtual switches, and T is the number of tables per switch.

By implementing true hardware virtualization, Corsa uses innovative techniques for collapsing the program memory space required for NxT tables. This involves the use of common code for tables that are the same or similar across different VSIs. As well, a hardware based context switching capability allows packets to be processed in different VSIs through a shared program memory space with zero performance overhead for context switching. This is essential to maintaining line rate packet processing in the presence of virtualization.

3.1.2. Size of OpenFlow tables

In order to multiply the amount of active state by NxT for large tables such as a 1 Million entry FIB, Corsa implemented a novel algorithmic lookup solution in hardware for longest prefix match (LPM). This allows the hardware switches to virtualize up to 8 full Internet routers holding a total of 8 Million unique addresses, rather than the naive assumption that a single 1Million entry TCAM would be reduced to 1Million/8 = 125 thousand LPM entries per VSI. If this were the case, despite the benefits of virtualization, the final solution would not be usable for simple Internet routing use cases.

3.1.3. Virtualization of QoS, metering and statistics

A non virtualized OpenFlow switch provides QoS facilities that scale with the number of physical ports. Traffic shaping, policing and queueing is typically performed on a per port basis, where typical port counts might range from 8 to 64 ports per hardware switch [4]. Logical OpenFlow switches on the other hand can have a large number of logical ports, as well the total number of ports that need to be traffic engineered is of the order NxP, where N is the number of logical switches, and P is the number of logical ports per logical switch. This product can easily reach numbers as large as 10,000, which is two orders of magnitude larger than in a non virtualized scenario.

In support of true hardware virtualization, the Corsa implementations use specialized ASIC based traffic engineering that can support the scale needed for implementing hierarchical QoS, metering and accurate statistics collection for each VSI.

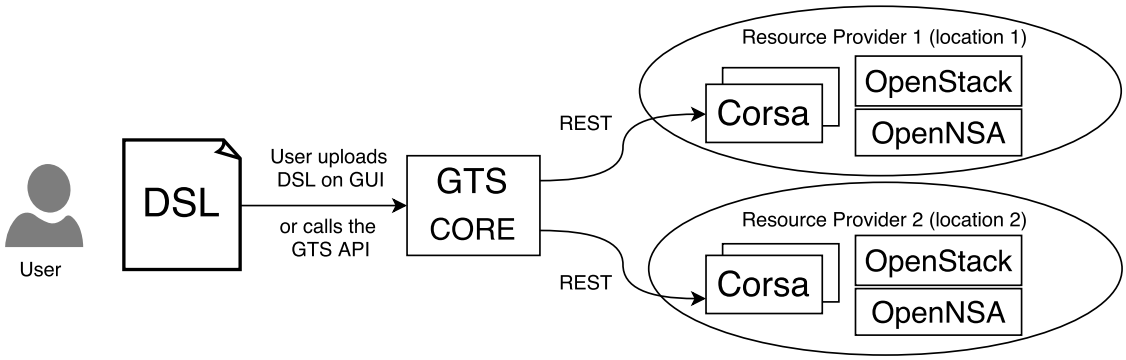

Figure 3: GTS high level architecture

4. Software Integration

To compliment the hardware advances presented in the previous Section, we also propose a platform-independent Domain Specific Language (DSL) that can be used to describe VSIs. The DSL is designed to be minimal and human-readable, yet it contains all necessary information for a virtual OpenFlow instance. In Figure 3, we are presenting a high-level view of the GTS system. As shown, users can request resources by uploading their DSL file on the GUI or by calling the GTS API with the DSL file.

The proposed description language specifies the number of ports required for the virtual switch along with their dataplane port identifiers as shown in lines #13 and #17. As presented in the port objects, the type of the ports attached to a virtual switch (control or data plane port) can be specified with the "mode" parameter - as show in line #22, where the port with id "CTRL" is specified to be a control port. This port will then be available using the IP address and subnet defined in lines #4 and #5. The controllers (primary and secondary), the DPID and the OpenFlow version of the switch are also clearly listed in lines #6 to #11. Apart from these compulsory parameters, one could add additional fields to this structure depending on the testbed. As shown in line #3 for instance, we have a location parameter in the DSL to select a virtual switch at a specified physical location. The "id" parameters for the VSI object and port objects are used to specify links between various objects in a broader resource description.

```
1 VSI {
2     id="OFX1"
3     location="COPENHAGEN"
4     switchIP="10.10.10.2"
5     switchSubnetMask="255.255.255.0"
6     switchDPID="0000000000000001"
7     controllerIP="10.10.10.100"
```

```
    controllerPort="6633"
    controllerIPSecondary="10.10.10.101"
    controllerPortSecondary="6633"
    OpenFlowVersion="OpenFlow13"
    port {
      ofport=1
      id="P1"
    }
    port {
      ofport=2
      id="P2"
    }
    port {
      id="CTRL"
      mode="CONTROL"
    }
}
```

5. Evaluation

In this Section we provide measurement results focusing on performance available through a single VSI and the performance isolation between multiple virtual OpenFlow switches created on the switch. The hardware and software components introduced in this paper have been set up at a lab environment at NORDUnet in Copenhagen. For all experiments we used a beta software/firmware version on the Corsa devices (version 1.4.0 build 18) that supports the VSI features, and a development version of GTS that integrates VSI support and the associated DSL extensions. On the Corsa device, we used the "Software-Defined Exchange" pipeline during the experiments, which is a multi-table OpenFlow 1.3 pipeline with 6 flow tables.

To generate traffic, we have utilized Dell PowerEdge R430 servers with Intel E5-2650L CPUs, 64GBs of RAM and a dual-port Intel x520 10Gbit/s network card. We ran DPDK-pktgen and used the Network Function Performance Analyser tool [5] for plotting results. The results presented here represent the performance that can be observed by GTS end users when using VSIs.

5.1. Throughput of a single VSI

We evaluated the throughput achievable through a single VSI. For these experiments, a single VSI with two virtual ports mapped to separate physical ports have been created. The two ports are connected to the DPDK traffic generator, where one is used to generate and one is used to receive traffic.

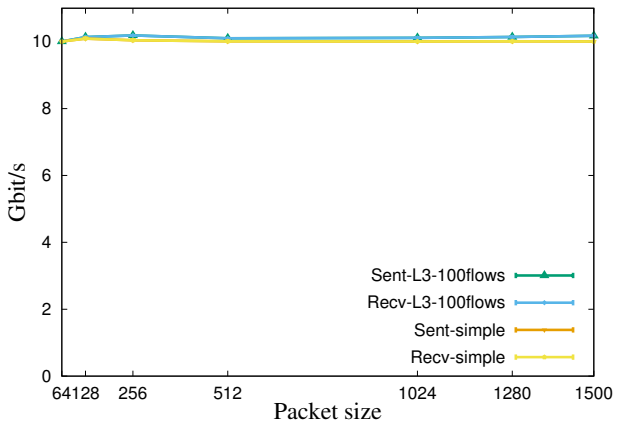

Figure 4: Result in Gbit/s

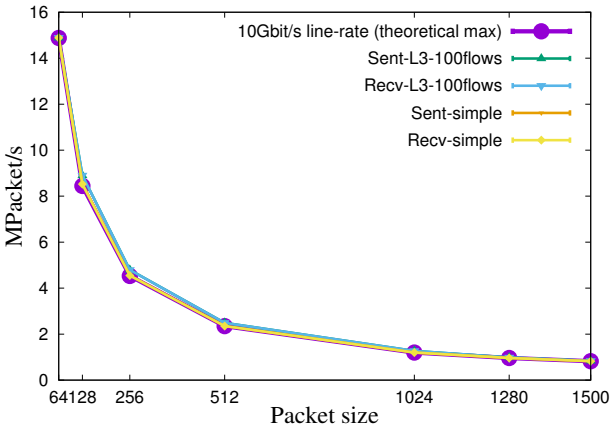

Figure 5: Result in MPacket/s

Two experiments were performed. First in the 'simple' experiment, we measured the throughput with different-sized packets and simple flow entries that match on input ports and forward on output ports. As for a second, more realistic '100flows' test, the L3 matching abilities of the device have been evaluated by using packets with randomly generated destination IPs (within a range of 100 IPs). In this case, we set the corresponding match entries on the device before the measurements using multiple flow tables and flow entries that match on the 100 destination IPs and forward traffic towards the receiver end of the packet generator.

As shown in Figure 4, VSIs produced line-rate performance during both 'simple' and '100flows' experiment with 10Gbit/s traffic. As presented from a different angle in Figure 5, the number of packets sent and received through the switch are close to the theoretical, line-rate limit, which is limited by the speed of the interfaces used.

5.2. Multiple VSIs sharing a physical port

We were also interested in the performance isolation properties of the VSIs, where multiple VSIs shared the same physical switch substrate. The generated traffic used two different VLAN tags that were set to direct traffic to the two different VSIs. The test setup is shown in Figure 6.

As expected, by not setting any restrictions or QoS features on the VSIs, both VSIs forwarded their 5Gbit/s traffic without any loss. This can be seen in Figure 7. In an other case, we set a limiter that drops packets at 3Gbit/s, but allows for a 5Mb burst. We set this limiter on the meter that is assigned to every packet arriving to VSI1. As a result, we could observer a drop in the received traffic to 3Gbit/s with larger packets and a slightly higher rate with small packets (due to burst allowed), as shown in Figure 7. Apart from simple limits and bursts, VSIs are equipped with more sophisticated packet coloring mechanisms to perform fine-grained policing.

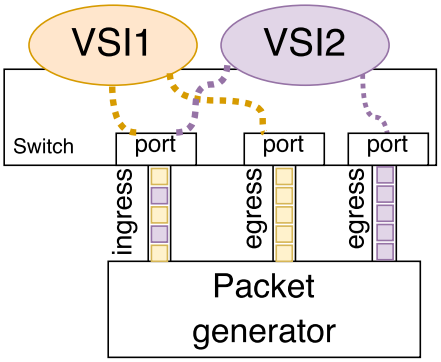

Figure 6: Two VSIs sharing the same physical ports

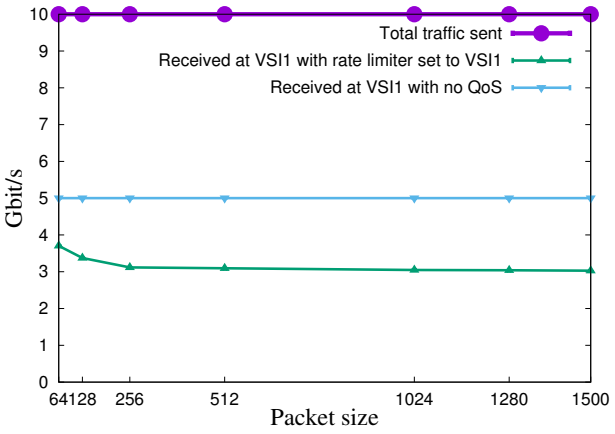

Figure 7: Results with multiple VSIs

6. Related work

In this section we are introducing other large-scale experimental networks and testbeds offering OpenFlow resources. We identify some differences between those and the VSI offerings introduced in this paper.

A prominent project, the Global Environment for Network Innovations (GENI) [6] is a [predominantly US] infrastructure supporting "at scale" research in computer networks, distributed systems and novel applications. The project is funded by the US National Science Foundation. The OpenFlow core of the GENI network consists of multiple interconnected OpenFlow-capable switches on both Internet2 and National LambdaRail (NLR) networks. The testbed operates several OpenFlow-compliant switches from various vendors located across the country. In their recent InstaGENI (An instant worldwide GENI Network) project, they rely on FlowVisor and an aggregate manager called the GENI Flowvisor OpenFlow Aggregate Manager (FOAM) [7] to provision experimental networks instantly. FOAM is also designed to enforce safety and isolation policies, such as limiting the switch CPU and memory resources available to each slice. In comparison to GTS, by VSIs, we can now provide higher switching performance and better isolation between switch slices than using software to perform these functions.

As of October 2012, Internet2 provides a nationwide high-speed (100Gbit/s) software defined network called the Advanced Layer2 Service (AL2S). The deployment includes routers of the Brocade MLX family and related Brocade NetIron platforms supporting OpenFlow, as well as Juniper Networks MX Series routers [8]. AL2S employs VLAN slicing across the SDN testbed to assign network subspaces to SDN applications or projects. VLAN slicing of this sort requires applications to know their assigned space and to stay within it. This poses security issues that are addressed by the Flowspace firewall proxy controller that inspects, authorizes, and updates the

backbone switches with each user flowspec. Flowspace firewall was an early innovation developed to allow sharing of the SDN switch infrastructure. The VSI concept described in this paper addresses many issues that have been exposed in the AL2S early experimental infrastructure.

The Energy Science Network (ESnet) [9] has also deployed different OpenFlow testbeds during the last few years. As stated on their website, their current 100G testbed consists of a dedicated 100G wave and a 10G overlay between Denver, Washington DC, New York, Atlanta, Amsterdam, and Geneva. Their testbed also consists of Corsa devices, as well as other, older OpenFlow 1.0 switches [10]. The ESnet Corsa devices have not yet been equipped with the Virtual Switch Instance feature, but this feature could easily be enabled in the near future. A supporting configuration and resource management environment - such as the GTS software suite - will need to be deployed to support efficient use of the VSIs.

A European program is the Future Internet Research and Experimentation (FIRE) initiative that offers various experimental network testbeds and projects [11]. Many of the research programs under FIRE offer access to SDN facilities, but these typically employ similar slicing processes - VLAN slicing, device scheduling, or promote complementary virtualization concepts such as NFV or NSC, and are therefore do not address the provisioning of global SDN-capable infrastructure environments The VSI concept, if made available globally, will enable many higher level or finer-grained experiments to be predicated on a truly global level.

A related, ongoing activity is the FED4FIRE [12] project that federates many testbeds with various OpenFlow resource offerings that usually either apply a software virtualization layer (FlowVisor) on top of physical devices or provide an entire switch to a user. The resources in FED4FIRE are allocated using the GENIv3 API with the RSpec data model. In comparison with GTS, we provide hardware virtualized switches without a software proxy and allow users to describe an OpenFlow resource using a human readable DSL. Efforts are underway to interwork the GTS Generalized Virtualization Model API and the GENIv3 API, SFA, and other experimental provisioning protocols.

AARNet, Australia's Academic and Research Network has also recently announced their Australia Wide-Area SDN Testbed, established in collaboration with nine universities and CSIRA Data61 [13]. Their infrastructure consists of four interconnected NoviFlow OpenFlow-enabled switches at AARNet backbone sites in Sydney, Melbourne, Perth, Seattle, all controlled by an ONOS controller. In comparison to GTS, AARNet's testbed provides the ONOS platform for experimenters, while GTS provides VSIs that can be used with any controller chosen by the users.

In the recent years, the GTS system has been deployed independently in networks outside GEANT's core. One of the prominent examples is the deployment of NORDUnet, named the Global Virtualized Service (GVS). GVS consists of pods located in Copenhagen (Denmark), Geneva (Switzerland), Miami (US) and Washington DC (US)

and it was the first network to test the VSI features. NORDUnet and GEANT are close collaborators, so there is a smooth and open technology sharing and cooperation between these two service providers.

7. Conclusion

This paper introduced the Virtual Switch Instances (VSI)s that provide virtual, yet high-performance OpenFlow switches for experimental network testbeds that allow users to specify all aspects of an OpenFlow switch (OpenFlow version, DPID, port identifiers, etc). The presented advances include software and hardware additions to the GEANT Testbed Service (GTS) project. We highlighted a DSL that can be used to describe virtual OpenFlow resources and presented an OpenFlow switch that allows the creation of VSIs. The evaluation results show high forwarding performance and isolation properties between VSIs.

Acknowledgements

NORDUnet[1] is the network consortium binding the Nordic Research and Education Networks of Sweden (SUNET), Norway (UNINETT), Finland (FUNET), Iceland (RHnet), and Denmark (DeiC).

The GEANT Network Project [2] is a joint effort of the European Commission and 35 National Research and Education Networks throughout Europe.

Corsa Technologies is an SDN switch developer based in Ottawa, CA. Corsa is focused on the service provider challenges and requirements for SDN in high performance core networks.

Mr Cziva was partially funded by NORDUnet and the Scottish Informatics and Computer Science Alliance (SICSA).

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: enabling innovation in campus networks, ACM SIGCOMM Computer Communication Review 38 (2) (2008) 69–74.

[2] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, Flowvisor: A network virtualization layer, OpenFlow Switch Consortium, Tech. Rep (2009) 1–13.

[3] S. Jouet, R. Cziva, D. P. Pezaros, Arbitrary packet matching in openflow, in: 2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR), IEEE, 2015, pp. 1–6.

[4] B. Sonkoly, A. Gulyas, F. Nemeth, J. Czentye, K. Kurucz, B. Novak, G. Vaszkun, On qos support to ofelia and openflow, in: 2012 European Workshop on Software Defined Networking, IEEE, 2012, pp. 109–113.

[1] www.nordu.net

[2] www.geant.net

[5] L. Csikor, M. Szalay, B. Sonkoly, L. Toka, NFPA: Network function performance analyzer, in: IEEE Conference on Network Function Virtualization and Software Defined Networks Demo Track (NFV-SDN), San Francisco, CA, USA, 2015, pp. 17–19, results are browsable at http://ios.tmit.bme.hu/nfpa.

[6] C. Elliott, Geni: exploring networks of the future (2009).

[7] N. Bastin, A. Bavier, J. Blaine, J. Chen, N. Krishnan, J. Mambretti, R. McGeer, R. Ricci, N. Watts, The instageni initiative: An architecture for distributed systems and advanced programmable networks, Computer Networks 61 (2014) 24–38.

[8] Internet2 to include brocade and juniper technologies in first 100g open, national-scale, software-defined network (2012).
URL https://lists.internet2.edu/sympa/arc/i2-news/2012-07/msg00002.html

[9] Energy science network.
URL http://es.net

[10] Energy science network - 100g testbed architecture.
URL http://www.es.net/assets/RD/Testbed/Testbed-Topology.pdf

[11] A. Gavras, A. Karila, S. Fdida, M. May, M. Potts, Future internet research and experimentation: the fire initiative, ACM SIGCOMM Computer Communication Review 37 (3) (2007) 89–92.

[12] Fed4fire - federation for future internet research and experimentation.
URL http://www.fed4fire.eu/

[13] Aarnet launches sdn innovation platform for researchers (2016).
URL http://news.aarnet.edu.au/aarnet-launches-sdn-innovation-platform-for-researchers/