An SDN-based Architecture for Network-as-a-Service

Mani Prashanth Varma Manthena*, Niels L. M. van Adrichem*, Casper van den Broek† and Fernando Kuipers*

*Network Architectures and Services, Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
{M.P.V.Manthena@student., N.L.M.vanAdrichem@, F.A.Kuipers@}
tudelft.nl

†ICT - Service Enabling and Management, TNO
Postbus 5050, 2600 GB Delft, The Netherlands
casper.vandenbroek@tno.nl

***Abstract*—Network-as-a-Service (NaaS) is a cloud-based service model that offers on-demand network connectivity and the provisioning and management of network services. However, the actual orchestration of dynamically allocating underlying resources to customer requirements is not trivial.**

In this paper, we propose an SDN-based approach to support the NaaS model. We implement a proof-of-concept (PoC) on a physical testbed and validate it through experimental performance evaluation.

I. Introduction

Network-as-a-Service (NaaS) is a cloud-based service model that offers on-demand network connectivity and the provisioning and management of network services. However, the actual orchestration of dynamically allocating underlying resources to customer requirements is not trivial. Currently, several service providers and network operators implement virtual overlay networks by using tunneling, tagging and labeling protocols such as GRE, IP-in-IP, V(X)LAN and MPLS. Although this type of network virtualization may enable NaaS by provisioning customizable network services, it does not change the main characteristics of the network. Although it realizes the short-term benefits of network virtualization, it adds new layers of complexity to the already complex network.

In this paper, we propose an architecture for service provider networks using SDN as its key virtualization enabling technology to enable NaaS. Moreover, the proposed architecture uses network monitoring technologies for more scalable, reliable and granular closed-loop network control and management. Our approach involves an implementation that facilitates incremental deployment by using SDN capable edge switches, while maintaining an existing MPLS-based core network.

We first discuss related work in section II. In section III, we introduce and discuss our architecture. In section IV, we demonstrate a Proof-of-Concept implementation on a physical testbed. Section V presents the performance evaluation of the PoC physical network testbed and an analysis and discussion of results. Finally, section VI concludes this paper.

II. Related Work

Duan et al. [1] present a comprehensive survey on how service-oriented architecture (SOA) principles can be applied to network virtualization. As a development in this direction, the authors in [2] present a framework that integrates the NaaS paradigm by abstracting the SDN control plane to implement a high-level network service orchestration model based on SOA principles. Similarly, Bueno et al. [3] propose a software framework based on NaaS and OpenFlow along with dynamic network configuration and status monitoring to enable on-demand, customizable and application-aware network services with end-to-end QoS guarantees. However, none of these approaches have been validated by a proof of concept implementation on a physical network testbed.

On the other hand, Gouveia et al. [4] propose a framework that realizes the full-stack implementation of OpenFlow based SDN, which is validated by a network testbed implementation and performance evaluation. However, this framework is difficult to implement on already existing network architectures, as it does not consider a gradual implementation strategy.

Casado et al. [5] present the major drawbacks of an OpenFlow based SDN architecture in terms of complexity in network address mapping and evolutionary inflexibility of the network. They propose a hybrid approach that retrospectively applies the insights of MPLS and distinction between network core and edge to overcome those major drawbacks. However, this hybrid approach does not explicitly consider a deployment scenario in existing networks and lacks closed-loop monitoring.

III. SDN-based Architecture

We take an evolutionary approach that, instead of upgrading the whole network at once, facilitates an incremental deployment scenario. We first present and discuss the data plane considerations in subsection III-A, followed by the control and management plane considerations in subsection III-B.

A. Data Plane Considerations

In Figure 1, a typical single domain service provider network is shown. In the figure, provider and provider edge routers are labeled as P and PE respectively. Similarly, customer edge routers are labeled in the figure as CE. In general, provider routers are MPLS switch routers and provider edge routers are MPLS edge routers, which are labeled in the figure as LSR and LER respectively.

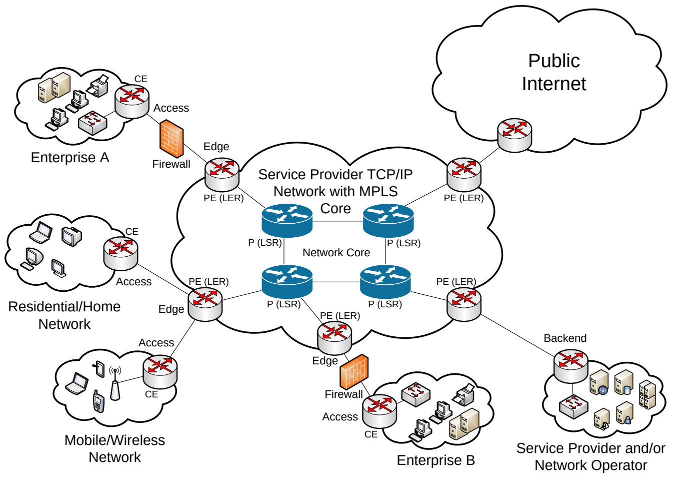


Fig. 1: A typical single domain service provider network.

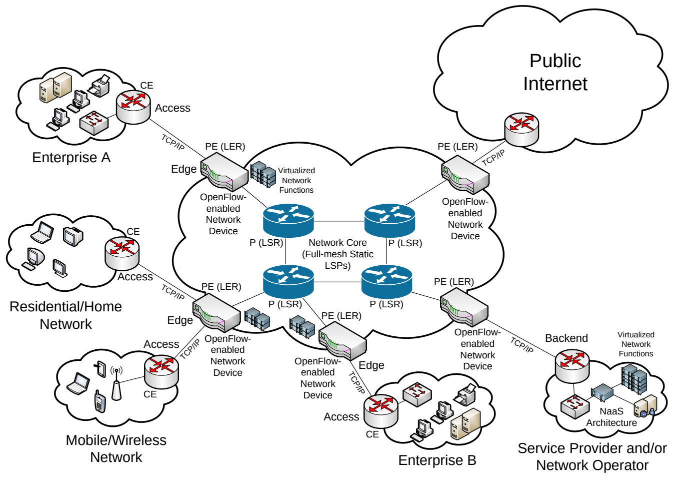

Public Internet
Enterprise A
TCP/IP
Virtualized Network Functions
OpenFlow-enabled Network Device
Network Core (Full-mesh Static LSPs)
Residential/Home Network
Mobile/Wireless Network
Enterprise B
Backend
NaaS Architecture
Service Provider and/or Network Operator

Fig. 2: Our proposed SDN-based architecture.

Since MPLS is currently widely implemented in service provider networks, we propose to only replace or update the edge devices with OpenFlow-enabled network devices while keeping the network core unchanged. The latest versions of the OpenFlow protocol and their switch specifications support key MPLS operations. Our legacy network core involves proactive installation of full-mesh static LSPs, in contrast to dynamic LSPs built through signalling and routing protocols. As a result, the network edge performs all complex network operations and functions on the incoming network traffic and steers it across the simple and static legacy network core by label switching.

In order to perform closed-loop network control and management, we propose to use sFlow [6] at the network edge and SNMP [7] at the network core. sFlow is chosen because most of the commercially available OpenFlow-enabled network devices, such as Open vSwitch [8], support sFlow. Similarily, we use SNMP because of its widespread adoption.

The mentioned data plane considerations are depicted in Figure 2. This approach results in an intelligent and complex network edge that is decoupled from a simple and static legacy network core.

B. Control Plane Considerations

Our control plane considerations primarily aim to provision simple abstractions of the underlying network resources, enabling network virtualization through SDN to the northbound NaaS based network service orchestration platform as proposed in [1][2][3]. For the complex network edge, the control plane involves (1) an OpenFlow based SDN controller for applying dynamic and flow-level traffic control, (2) an sFlow based network analyzer for flow-level traffic monitoring and (3) a network configuration system that configures the underlying resources. Furthermore, for NaaS based network service orchestration and closed-loop network control, the OpenFlow based SDN controller and sFlow based network analyzer must expose their northbound APIs. For the simple and static legacy network core, one can reuse the current legacy network management system while exposing SNMP through an analyzer's API.

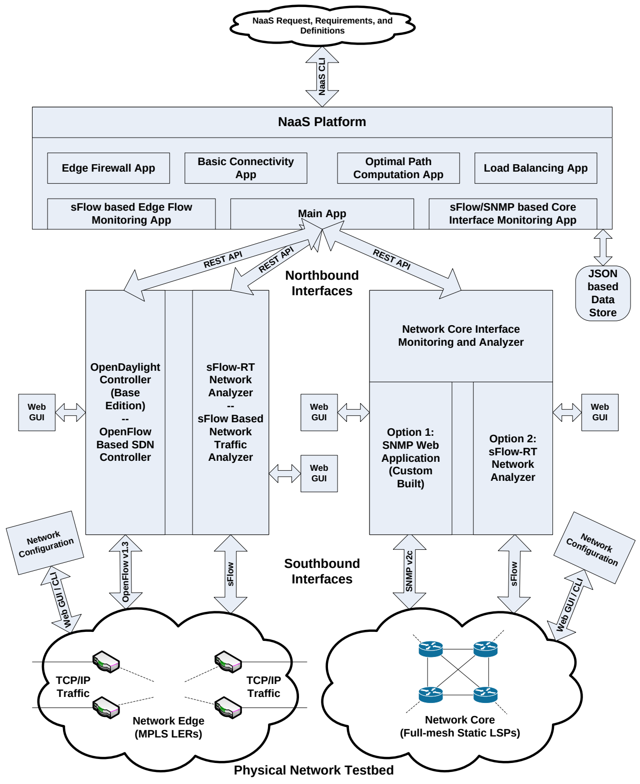

NaaS Request, Requirements, and Definitions
NaaS CLI
NaaS Platform
Edge Firewall App
Basic Connectivity App
Optimal Path Computation App
Load Balancing App
sFlow based Edge Flow Monitoring App
Main App
sFlow/SNMP based Core Interface Monitoring App
JSON based Data Store
REST API
REST API
REST API
Northbound Interfaces
OpenDaylight Controller (Base Edition) -- OpenFlow Based SDN Controller
sFlow-RT Network Analyzer -- sFlow Based Network Traffic Analyzer
Network Core Interface Monitoring and Analyzer
Option 1: SNMP Web Application (Custom Built)
Option 2: sFlow-RT Network Analyzer
Web GUI
Network Configuration
Web GUI / CLI
OpenFlow v1.3
sFlow
Southbound Interfaces
SNMP v2c
sFlow
TCP/IP Traffic
Network Edge (MPLS LERs)
Network Core (Full-mesh Static LSPs)
Physical Network Testbed

Fig. 3: The Proof-of-Concept design of our architecture.

IV. Proof of Concept

In this section, we first present and discuss our Proof of Concept design in subsection IV-A, followed by its implementation on a physical network testbed in subsection IV-B.

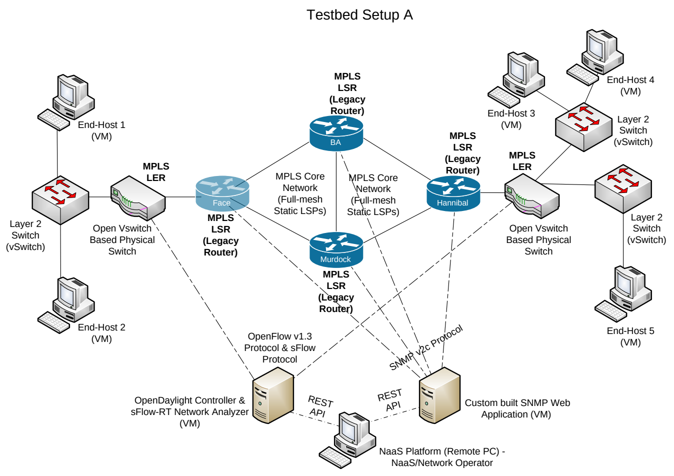

Testbed Setup A
MPLS LSR (Legacy Router)
BA
End-Host 1 (VM)
End-Host 2 (VM)
End-Host 3 (VM)
End-Host 4 (VM)
End-Host 5 (VM)
Layer 2 Switch (vSwitch)
Layer 2 Switch (vSwitch)
Layer 2 Switch (vSwitch)
MPLS LER
Open Vswitch Based Physical Switch
Face
MPLS LSR (Legacy Router)
MPLS Core Network (Full-mesh Static LSPs)
MPLS Core Network (Full-mesh Static LSPs)
MPLS LSR (Legacy Router)
Hannibal
MPLS LER
Open Vswitch Based Physical Switch
Murdock
MPLS LSR (Legacy Router)
OpenFlow v1.3 Protocol & sFlow Protocol
SNMP v2c Protocol
OpenDaylight Controller & sFlow-RT Network Analyzer (VM)
REST API
REST API
Custom built SNMP Web Application (VM)
NaaS Platform (Remote PC) - NaaS/Network Operator

Fig. 4: Testbed Setup A with Juniper (M10i) legacy routers as the network core MPLS LSRs.

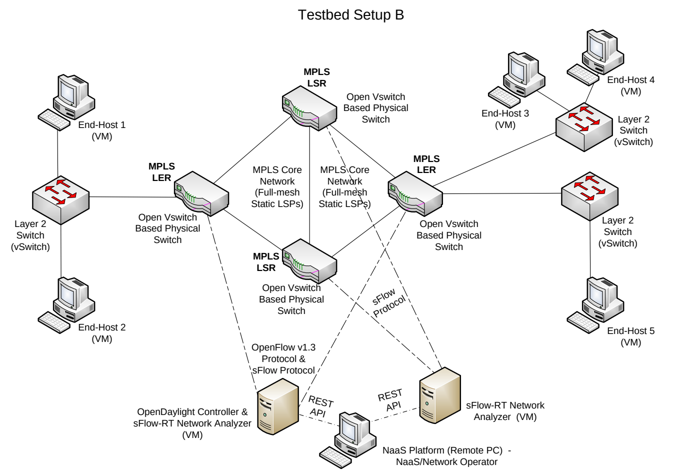

Testbed Setup B
MPLS LSR
Open Vswitch Based Physical Switch
End-Host 1 (VM)
End-Host 2 (VM)
End-Host 3 (VM)
End-Host 4 (VM)
End-Host 5 (VM)
Layer 2 Switch (vSwitch)
Layer 2 Switch (vSwitch)
Layer 2 Switch (vSwitch)
MPLS LER
Open Vswitch Based Physical Switch
MPLS Core Network (Full-mesh Static LSPs)
MPLS Core Network (Full-mesh Static LSPs)
MPLS LER
Open Vswitch Based Physical Switch
MPLS LSR
Open Vswitch Based Physical Switch
sFlow Protocol
OpenFlow v1.3 Protocol & sFlow Protocol
OpenDaylight Controller & sFlow-RT Network Analyzer (VM)
REST API
REST API
sFlow-RT Network Analyzer (VM)
NaaS Platform (Remote PC) - NaaS/Network Operator

Fig. 5: Testbed Setup B with Pica8 (P-3290) open switches as the network core MPLS LSRs.

A. Proof of Concept Design

Our Proof of Concept (PoC) design is depicted in Figure 3. The PoC design does not involve any virtualized network functions being implemented on the underlying virtual servers. The NaaS based northbound service provisioning platform is accessed through CLI. The OpenDaylight Controller (Base Edition) [9] is chosen as the OpenFlow based SDN controller because of its full-stack support for SDN and NFV, well-documented northbound APIs and large developer community.

For monitoring, we use the sFlow-RT network analyzer [10] and we have built a custom SNMP based web application with northbound APIs for network core interface monitoring, exposing link failures and utilization events at customizable polling intervals and thresholds. Vendor-specific CLIs and web GUIs are used for network device configurations as our PoC design involves only static network configuration. The NaaS platform involves an application that acts as an abstraction layer to represent all underlying APIs as simple abstractions and function calls to other applications in the platform. The implementation of our PoC is published open source and can be found at our GitHub web page [11].

B. Implementation on a Physical Network Testbed

Our physical network testbed consists of 4 Pica8 open switches (2 P-3922 48-port 10-GBE and 2 P-3290 48-port 1-GBE switches), 4 Juniper M10i series routers, 7 Linux based virtual machines (VMs) running on 5 different servers and a remote PC. The 4 Pica8 open switches can either be used in legacy L2/L3 mode or Open vSwitch mode.

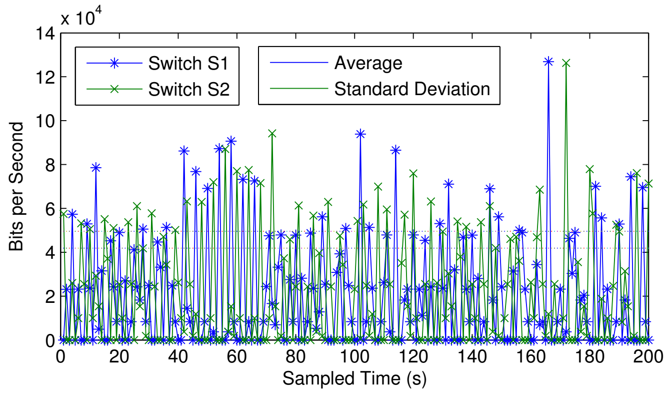
In our physical network testbed, we use one VM for running the OpenDaylight Controller and sFlow-RT network analyzer for edge flow monitoring, one VM for running our custom built SNMP web application and optionally the sFlow-RT network analyzer for core interface monitoring, 5 VMs running on the remaining 3 servers as our 5 end-hosts and a remote PC for hosting the NaaS platform. Moreover, end-host 5 is configured to be in a different IP subnet compared to the other end-hosts in the physical network testbed in order to emulate routing across service provider networks. At the network edge, we use the 2 P-3922 Pica8 open switches as the OpenFlow-enabled LERs. At the network core, we have two different implementations, one implementation involves the four Juniper M10i series routers as the legacy network core LSRs, whereas the other implementation involves the two P-3290 Pica8 open switches as the static network core LSRs. In the latter, we use sFlow instead of SNMP for network core interface monitoring in the comparative testbed setup as the Pica8 open switches support sFlow. Thus, we have two testbed setups, which are respectively depicted as Testbed Setup A and Testbed Setup B in Figures 4 and 5.

In both testbed setups, network devices are connected through 1GB fiber-optic interfaces, while the end-host VMs are connected to the network through gigabit Ethernet cables. Furthermore, we pre-install full-mesh static LSPs performing penultimate hop popping in the network core of both our testbed setups. Finally, the OpenFlow-enabled LERs need to replace the destination MAC addresses of the incoming network traffic with that of the next-hop router.

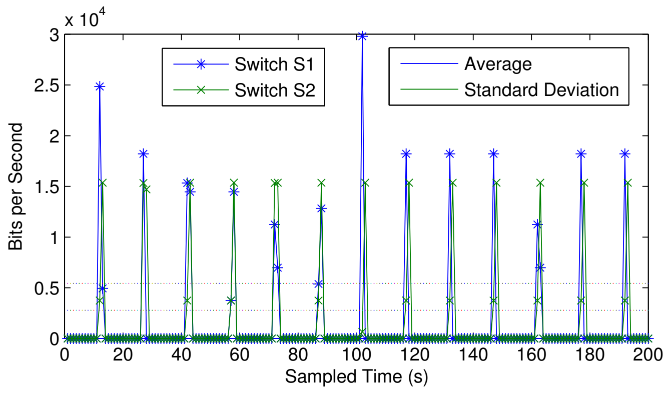
V. Performance Evaluation and Validation

In this section, we present the experimental performance evaluation of our PoC. We focus on performance analysis of the involved network control overhead. In order to perform our evaluation, we use the packet analyzer Wireshark at the involved VMs (controllers, analyzers, monitors, and end-hosts). Since the captured packet information from Wireshark involves time-stamped data, we resample this time series data into buckets of 1 second to plot network load in terms of frames per second, bytes per second, and bits per second over this sampled time.

For the performance analysis of the involved control overhead in the PoC physical network testbed, we first study and analyze the involved OpenFlow, sFlow, and SNMP protocol



(a) Testbed Setup A.



(b) Testbed Setup B.

Fig. 6: OpenFlow protocol traffic load samples at the OpenDaylight Controller in the PoC physical network testbed.

traffic in Testbed Setup A at the OpenDaylight Controller, sFlow-RT network analyzer, and custom built SNMP web application, respectively, while provisioning and managing the PoC basic network connectivity services over it. Later, a similar analysis is carried out in Testbed Setup B and compared with that of Testbed Setup A. In essence, Testbed Setup B only differs from Testbed Setup A in terms of its network core, it has open switches instead of legacy routers and uses sFlow instead of SNMP based network monitoring.

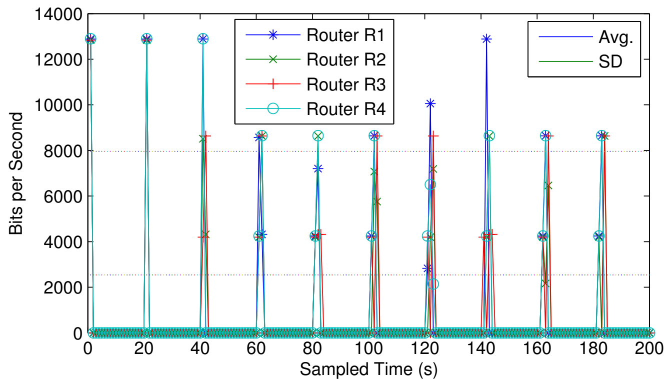
As can be seen from Figure 6, the overall network control overhead is much higher in Testbed Setup A compared to that of Testbed Setup B. At the network edge, the average value of the total OpenFlow protocol traffic load at the OpenDaylight Controller to/from the two underlying open switches in Testbed Setup A is around 45 Kbps with a standard deviation of around 50 Kbps, whereas in Testbed Setup B the average value is around 1.5 Kbps with a standard deviation of around 4.5 Kbps. This huge difference can be explained by analyzing their involved traffic load samples.

In general, the OpenDaylight Controller gathers statistics from the underlying open switches every 15 seconds, i.e. flow, group, meter, port counters, and table statistics, using the OpenFlow statistics request and reply messages, whose traffic peaks can be clearly noticed in Subfigure 6b. Furthermore, the only other major operation of the OpenDaylight controller in the PoC physical network testbed is that of installing and deleting flows in the underlying switches, which involves OpenFlow packets of around 200 bytes large. However, the reason for the high traffic load in Testbed Setup A is due to the involved link-state routing protocol, OSPF, in its legacy network core. The legacy routers send out multicast OSPF "Hello" packets every 8 seconds. The edge open switches, upon receiving those packets, send them to the OpenDaylight Controller for corresponding action, which in turn returns them to the underlying open switches to appropriately multicast them. In this manner, they cause a high traffic load in the network and at the OpenDaylight Controller.

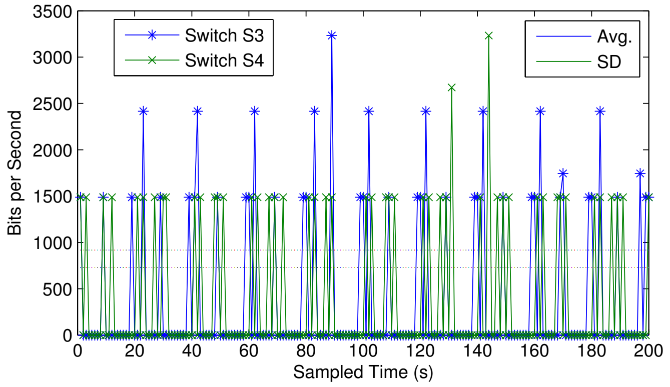
Similarly, our end-host VMs are installed with an LLDP (Link Layer Discovery Protocol) implementation daemon called *lldpd*, which sends out multicast LLDP requests every 30 seconds and similarly introduces high traffic load in the network. Moreover, the sFlow based network edge traffic monitoring involves sampling of packets and exporting of interface statistics to the sFlow-RT network analyzer at a configured rate. Thus, the sFlow-RT network analyzer is also over loaded with the additional sFlow protocol traffic in Testbed Setup A. Nevertheless, both LLDP and OSPF are additional features to the PoC physical network testbed and can be removed, as shown in Testbed Setup B and Subfigure 6b, or handled accordingly by the OpenDaylight Controller to avoid additional network control overhead.

As can be seen from Figure 7, the average value of the total SNMP protocol traffic load at the custom built SNMP web application to and from the four underlying legacy routers in the network core at Testbed Setup A is around 2.5 Kbps with a standard deviation of around 6 Kbps, whereas in Testbed Setup B the average value of the total sFlow protocol traffic load at the sFlow-RT network analyzer from the two underlying open switches is around 0.8 Kbps with a standard deviation of around 1 Kbps. This difference can be explained by analyzing their involved traffic load samples.

In Testbed Setup A, at the network core, the custom built SNMP web application gathers only three interface counters from the Management Information Bases, being ifOperStatus, ifInOctetes and ifOutOctetes in IF-MIB of SNMP MIB-2, in the underlying legacy routers at once every 20 seconds. The involved SNMP v2c request and reply packets in gathering the three interface counters are each of a size around 90 bytes. In Testbed Setup B, at the network core, the sFlow standard implementation in the underlying open switches is configured with a packet sampling rate of 1000 and an interface statistics polling rate of 20 seconds. The involved sFlow packets in exporting the per-interface statistics and sampled packets are each, respectively, around 186 bytes and 218 bytes large. Furthermore, two or more interface statistics message and sampled packets may be exported as a single sFlow packet,



(a) SNMP protocol - Testbed Setup A.



(b) sFlow protocol - Testbed Setup B.

Fig. 7: Network core monitoring traffic load samples in the PoC physical network testbed.

in which case the total packet size is less than the sum of the individual packet sizes when sent seperately. Protocol traffic load measurements in the corresponding testbed setups are shown in Subfigures 7a and 7b, respectively.

Although Testbed Setup B has less network core devices compared to that of Testbed Setup A, the control overhead at the network core is still logically higher in Testbed Setup A compared to that of Testbed Setup B. Firstly, SNMP based network monitoring in Testbed Setup A involves pull-based gathering of only three interface counters for detecting interface failure and high utilization events, whereas the sFlow based network monitoring in Testbed Setup B involves push-based exporting of all the interface statistics from the underlying open switches. Secondly, sFlow based network monitoring additionally involves flow sampling for more visibility into the traffic at the network core. Finally, SNMP based network monitoring gathers interface statistics of the underlying legacy routers once every 20 seconds, whereas sFlow based network monitoring involves exporting of interface statistics once every 20 seconds for each interface.

Additionally, we have found that Testbed Setup B experiences a higher round-trip delay compared to that of Testbed Setup A. The main reason behind this is that the two open switches in Testbed Setup B's network core execute MPLS forwarding operations in their CPU instead of their hardware ASICs. Thus, the involved open switches in our proposed evolutionary approach must support MPLS switching operations in their hardware ASICs for much faster switching, avoidance of network bottlenecks, and to prevent unnecessary load on the switch CPUs. Altogether, our proposed evolutionary approach performs much better compared to the legacy solutions in terms of network control overhead as it avoids the usage of legacy signaling and control protocols. Furthermore, the incremental deployment of open switches in the network core will greatly reduce the network control overhead while enabling much better visibility into the network core. Thus, our proposed evolutionary approach enables flexible and independent evolution of both the network core and edge.

VI. Conclusion

In this article, we have proposed an SDN-based architecture for the Network-as-a-Service (NaaS) cloud-based service model. We have implemented a Proof-of-Concept implementation on a physical network testbed, combined with provisioning and management of basic network connectivity services over it. Our performance evaluation yields sufficient results with low control overhead and a quick response time.

References

[1] Q. Duan, Y. Yan, and A. V. Vasilakos, "A survey on service-oriented network virtualization toward convergence of networking and cloud computing," *Network and Service Management, IEEE Transactions on*, vol. 9, no. 4, pp. 373–392, 2012. II, III-B

[2] Q. Duan, "Network-as-a-service in software-defined networks for end-to-end qos provisioning," in *Wireless and Optical Communication Conference (WOCC), 2014 23rd*, May 2014, pp. 1–5. II, III-B

[3] I. Bueno, J. I. Aznar, E. Escalona, J. Ferrer, and J. A. Garcia-Espin, "An opennaas based sdn framework for dynamic qos control," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*. IEEE, 2013, pp. 1–7. II, III-B

[4] R. Gouveia, J. Aparicio, J. Soares, B. Parreira, S. Sargento, and J. Carapinha, "Sdn framework for connectivity services," in *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 3058–3063. II

[5] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: a retrospective on evolving sdn," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 85–90. II

[6] P. Phaal, S. Panchen, N. McKee, "Inmon corporation's sflow: a method for monitoring traffic in switched and routed networks," RFC 3176 (INFORMATIONAL), Internet Engineering Task Force, Sep. 2001. [Online]. Available: https://www.ietf.org/rfc/rfc3176.txt III-A

[7] D. Harrington, R. Presuhn, B. Wijnen, "An architecture for describing simple network management protocol (snmp) management frameworks," RFC 3411 (INTERNET STANDARD), Internet Engineering Task Force, Dec. 2002, updated by RFCs 5343, 5590 and Obsoletes RFC 2571. [Online]. Available: http://www.ietf.org/rfc/rfc3411.txt III-A

[8] "Open vswitch: an open virtual switch," GitHub Web Front-End and Project, Open vSwitch. [Online]. Available: http://git.openvswitch.org III-A

[9] "Opendaylight "hydrogen" base edition," Linux Foundation Collaborative Project, OpenDaylight. [Online]. Available: https://wiki.opendaylight.org/view/Release/Hydrogen/Base/User_Guide IV-A

[10] "sflow-rt: sflow based network analyzer," InMon Corp. [Online]. Available: http://www.inmon.com/products/sFlow-RT.php IV-A

[11] M. P. V. Manthena, "Tudelftnas/sdn-naasplatform," Mar. 2015. [Online]. Available: https://github.com/TUDelftNAS/SDN-NaaSPlatform IV-A