Control-Plane Slicing Methods in Multi-Tenant Software Defined Networks

C. Argyropoulos, S. Mastorakis†, K. Giotis, G. Androulidakis, D. Kalogeras, V. Maglaris
Network Management & Optimal Design Laboratory (NETMODE)
School of Electrical & Computer Engineering
National Technical University of Athens (NTUA), Greece
{cargious, coyiotis, gandr, dkalo, maglaris}@netmode.ntua.gr, mastorakis@cs.ucla.edu

***Abstract*— In this paper, we focused on two prevailing architectural approaches for control-plane virtualization in multi-tenant OpenFlow-ready SDN domains: The first permits the delegation of a specific, non-overlapping part of the overall flowspace to each tenant OpenFlow controller, exposing him/her the entire substrate topology; the second conceals the substrate topology to tenants by abstracting resources and exposing user-controlled (tenant) Virtual Networks (VNs). For both cases, we propose and analyze three control-plane slicing methods (domain, switch and port-wide), enforced by the management plane, that safeguard control-plane isolation among tenant VNs. Their effectiveness is assessed in terms of control-plane resources (number of flowspace policy rule entries, table lookup times and memory consumption) via measurements on a prototype implementation. To that end, we introduced and prototyped the Flowspace Slicing Policy (FSP) rule engine, an automated mechanism translating substrate management-plane policies into VN mapping control-plane rules. Our experiments, involving thousands of tenants VN requests over a variety of WAN-scale network topologies (e.g. Internet2/OSE3 and GÉANT), demonstrate that the port-wide slicing method is the most efficient in terms of tenant request acceptance ratio, within acceptable control-plane delays and memory consumption.**

Keywords— Software Defined Networking (SDN); OpenFlow; Network Virtualization; Multi-tenancy; Control-Plane Slicing; Flowspace Slicing Policy

I. INTRODUCTION

Dynamic multi-tenant, multi-site cloud service provisioning is expected to profit from the adoption of Software Defined Networking (SDN) technologies. SDN can leverage coexistence of multiple Virtual Networks (VNs) that, apart from data-plane isolation, empowers VN owners (tenants) with delegated control-plane functionality.

Large-scale multi-tenant, multi-site, cloud environments, interconnected by WANs, increasingly rely on network virtualization technologies and protocols. In such environments, each tenant can request his/her own dedicated network slice consisting of logical and physical network resources, shared across one or more networking substrates, applying custom forwarding logic and developing advanced services within his/her slice (VN), without being aware of the physical network substrate (infrastructure).

The key-feature of SDN is the decoupling of control and data-plane, as demonstrated in OpenFlow (OF) architectures [4]. OF established a standardized vendor-independent interface between a centralized control-plane schema (OF Controller - OFC) and a number of distributed data-plane entities (OF-enabled switches). Leveraging on OpenFlow control-plane architectures, large cloud providers began to develop proprietary Software Defined Networks (SDNs) to create production-quality, global-scale, single-tenant control-plane architectures, e.g. Google's internal WAN [5] and NTT's Enterprise Cloud [6].

Despite the aforementioned advances in control-plane technologies, the problem of efficient substrate slicing amongst multiple tenants remains an open research issue. The challenge for multi-tenant service provisioning is to export control-plane functionalities to multiple tenants in order to create isolated scalable virtual topologies with virtual routers, switches and firewalls. To that end, the control-plane needs to be segmented and associated with subsets of flows, exclusively controlled by different tenants, without introducing considerable management overhead.

In this context, our work intends to identify and evaluate slicing policies that segment the flowspace (i.e., the set of flows that constitute virtual networks owned by tenants) while enforcing isolation amongst a large number of tenant-controlled virtual networks. Notably:

- We analyze two approaches to implement multi-tenant SDN architectures that enhance control-plane delegation to tenants (slice owners).
- We define three, compatible with the aforementioned SDN architectures, control-plane slicing methods (domain-wide, switch-wide and port-wide) that allow delegation of easily managed flowspace segments.
- We implement the Flowspace Slicing Policy (FSP) rule engine, a mechanism that translates infrastructure management-plane slicing policies into VN mapping control-plane rules used for benchmarking the three slicing methods.
- We evaluate the performance of the proposed slicing methods over diverse real network topologies, e.g. the U.S Academic backbone Internet2/OSE3 [7] and the

† Currently with the Computer Science Department, UCLA, Los Angeles, California, U.S.A

This work was partially supported by the European Commission, 7th Framework Program for Research and Technological Development, Grant Agreement No. 605243 (GN3plus).

pan-European Research & Education Network GÉANT [8], in terms of control-plane resources using the FSP engine.

- We evaluate the feasibility of the proposed slicing methods in terms of introduced time overhead and memory consumption by injecting flowspace rules produced by the FSP engine into a popular Proxy Controller implementation, the FlowVisor [9].

The remainder of our work is organized as follows: In Section II, we summarize related work for multi-tenant SDNs. In Section III, we analyze candidate-sharing architectures for multiple virtual network creation on top of a common infrastructure, define three control-plane slicing methods and study their performance. In Section IV, we describe our experimental evaluation methodology and present experimental results. Finally, in Section V, we report on the conclusions of our work and refer to open issues for future research.

II. Related Work

The first effort of slicing the control-plane of OpenFlow-enabled networks was FlowVisor [9], a special-purpose transparent Proxy Controller that slices substrate resources and provides data forwarding programmability to multiple tenants. The FlowVisor partially supports the concept of network virtualization, as it does not allow network topology abstractions needed, for example, for path migration and path splitting functions [10]. To that end, we used FlowVisor as an option of our experimental comparative evaluations of the slicing methods. A promising undergoing Proxy Controller effort is the Flowspace Firewall [11]. Its design focuses on high performance restricting slicing capabilities only to the data link layer. The presenting slicing methods are compatible with data link layer slicing and thus can be applied to an environment that will use the Flowspace Firewall.

The efficient mapping of virtual nodes and links of a Virtual Network (VN) request to substrate network resources, commonly referred as VN embedding (VNE) problem [12][13], is a prerequisite for efficient multi-tenancy. The VNE functionality should be handled within the network hypervisor virtualization component introduced in [14], as in the OpenVirtex platform [15] that not only slices the entire flowspace amongst the tenants, but also provides each tenant with an isolated instance of a fully virtualized network. The definition of virtual to physical mapping is specified by a set of rules to be applied to the infrastructure topology. In this paper we suggest OpenVirtex compliant methods for the creation of such rules that should be enforced to an infrastructure shared by multiple tenants. Furthermore, our multi-tenant SDN architecture, described in Section III, is compatible with network virtualization semantics, such as the link/node abstraction and Network Hypervisor described in [14], as well as the path splitting and migration presented in [10].

In [16], VeRTIGO, another network virtualization platform was presented, built on top of the Proxy Controller concept. This platform is based on FlowVisor, extending it with abstraction capabilities (different views of the network to different controllers). However, this work (including the virtual embedding algorithms for VeRTIGO platform [17]) is implicitly based on the assumption that each accepted VN request is efficiently translated to flowspace rules that must be established in the appropriate Proxy Controller. We contribute studying various mapping (slicing) methods and investigate the limitations of a Proxy Controller, which is used as an architectural component in [16], [17].

Multi-tenancy, as a use case in network virtualization, triggered the concept of shared infrastructure controllers. Notably, FlowN [18] is a multi-tenant OpenFlow controller aiming at extending NOX [19] in order to enable individual tenants to write arbitrary software with full control over a reserved flowspace. Our work, in essence, complements the FlowN approach by introducing slicing methods generic enough to permit the enforcement of a flowspace isolation policy for large-scale topologies.

Closely related to the robust and prompt functionality of a sliced control-plane in WAN environments is latency. Namely, the suboptimal location of control elements within a network infrastructure influences every aspect of a decoupled control-plane, from state distribution algorithms to fault tolerance and performance metrics. This problem is studied in [20], aiming at reducing the average or worst case latency between controllers and OF-enabled switches. Its main conclusion was that, for wide-area, large-scale footprints, a small number of well-placed controllers may form a robust SDN control-plane with acceptable latency. In this work, we extend the controller placement problem, taking into account processing delays introduced by the three multi-tenant slicing methods.

III. Leveraging OpenFlow/SDN for Control Plane Slicing

A. Proxy Controller Architecture

Flowspace delegation in SDNs can be implemented using an intermediate control-plane Slicing Layer, as shown in Figure 1. Every OpenFlow control message crossing the Slicing Layer is forwarded to a tenant OpenFlow controller (Tenant Layer) dictated by the flowspace slicing policy. Respectively, OpenFlow control messages, originated from a tenant OpenFlow controller (OFC), are intersected with the existing flowspace slicing policy defined within this intermediate level. The flowspace slicing policy is structured according to a control-plane slicing method, i.e., an algorithm, which guarantees the creation of non-overlapping flowspace rules. Control messages are routed via a transparent Proxy Controller (e.g. FlowVisor), as shown in Figure 1. In this architecture, the data-plane is defined across OpenFlow-enabled switches while the control-plane is sliced among tenant OpenFlow controllers.

The figure below adopts the well-known overlay principle of control and data-plane layers as well as a vertical Policy Based Network Management (PBNM) plane, consistent with reference models of various standardization bodies [21], [22], [23]. Namely, our proposed Flowspace Slicing Policy rule engine is the equivalent of a Policy Decision Point (PDP), which translates policy directives from a tenant to flowspace slicing policy rules. The Proxy Controller constitutes the Policy Enforcement Point (PEP) in this PBNM system [24], [25].

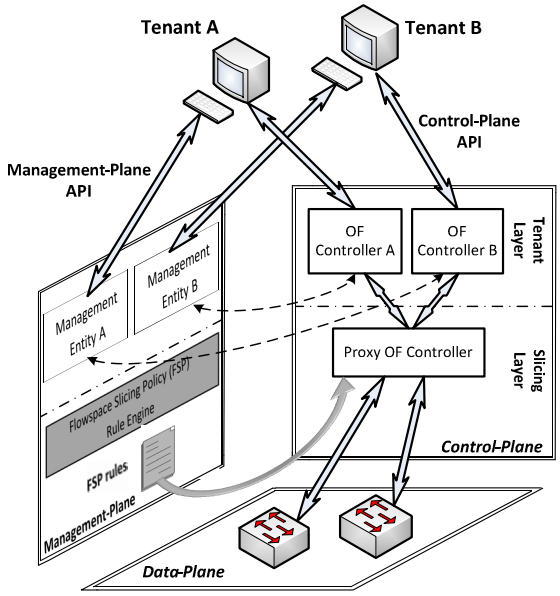


Figure 1. Proxy Controller-based architecture

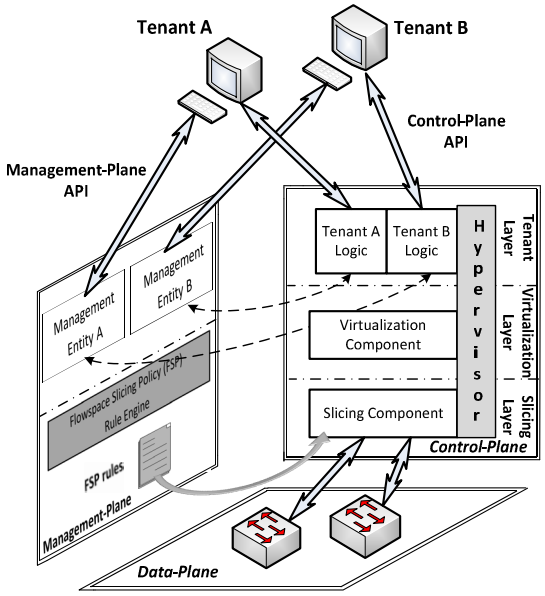


Figure 2. Network Hypervisor-based architecture

B. *Network Hypervisor Architecture*

A second multi-tenant SDN option is based on a Network Hypervisor, as shown in Figure 2. In this architecture, the control-plane consists of three distinct layers:

- The Slicing Layer, responsible for the creation of non-overlapping flowspace rules.
- The Virtualization Layer, responsible for the exposure of isolated virtual networks to tenant controllers. It may incorporate virtual network mapping algorithms responsible for link/node abstractions [14], path splitting and migration.
- The Tenant Layer, which implements the forwarding logic of the data-plane. This logic can be pre-defined by the infrastructure provider or, optionally, be the responsibility of users (tenants). In contrast with the Proxy Controller-based Architecture, the Tenant Layer of the Network Hypervisor co-hosts the forwarding logic of tenants.

Note that the virtualization layer can be absent, in which case each tenant can request his/her own flowspace with no network abstraction schema.

C. *Control-Plane Slicing Methods*

Multi-tenancy in both architectures requires classification of packets into flows, typically via logical separators within packet headers (packet ID). In this context, a shared infrastructure controller may consider a packet ID, preferably a L2 header field, as the identifier of a slice. Taking into consideration that the latest OpenFlow specification [26] supports multiple MPLS labels/ VLAN IDs, their use as packet IDs is possible.

Using packet IDs, e.g. <VLAN ID>, is a natural way to classify data-plane flows per slice, but may fail, if the number of slices exceeds the range of packet ID values (VLAN IDs are restricted to 4096 per domain). To account for this scalability problem, we considered logical separators that also specify data-plane resources pertaining to particular slices (tenants). Thus, a slice can be identified within an SDN controller via multiple logical separators, e.g. a tuple such as <VLAN ID, switch ID>, or <VLAN ID, switch ID, switch port ID>.

Following the aforementioned rationale, a network control-plane slicing method can be defined as an algorithm, which guarantees the creation of non-overlapping flowspace rules.

To that end, we experimented with the following three network slicing methods:

- **Domain-wide slicing.** Each network slice is strictly identified via a single value of the packet ID, using a unique logical separator per domain, e.g. <VLAN ID>
- **Switch-wide slicing.** Each slice is identified via multiple logical separators that also include the identification of the switching elements that it spans, e.g. < VLAN ID, switch ID>
- **Port-wide slicing.** Each slice is further identified with specific ports of the switching elements, e.g. < VLAN ID, switch ID, switch port ID>

An alternate approach would be to implement multiple VLAN ID push/pop actions per tenant VN slice, as implemented in MPLS backbones [27]. However, this would require multiple flowspace rules compared to the port-wide slicing method. As shown in Section IV, the port-wide slicing method requires a quite large number of flowspace rules. Thus, VLAN ID rewriting would lead to non-scalable solutions in terms of management complexity and control-plane resources, especially for WANs, studied in this paper.

D. Flowspace slicing policy

Regardless of the slicing method that an infrastructure provider would select, isolation among slices (tenants) within a common physical network must be enforced, as described in [9]. It is the responsibility of the shared infrastructure controller to create non-overlapping flowspace rules, thus enforcing isolation among tenant control-planes. In what follows, we present the implications of isolation enforcement for each of the aforementioned slicing methods.

In the domain-wide slicing, tenant should be prevented from reusing an already reserved separator instance (e.g. VLAN ID) across a domain.

Enforcing isolation in switch-wide slicing is more complex, as a logical separator instance (e.g. VLAN ID) can be used by multiple network slices across the physical topology risking conflicts within a VLAN. At an extreme, it may induce malicious poisoning of the control-plane by data-plane traffic. In Figure 3(a), we illustrate how a conflict can occur with tenant K and L slices allocated within two distinct switches A and B, which are interconnected via port 2 and port 1 respectively, but with the same separator VLAN i. If tenant K chooses port 2 of switch A as the egress port, packets would be forwarded to switch B, hence, tenant's L OpenFlow controller would be overloaded by alien OpenFlow control messages. As a result, port 2 for the specific logical separator must not be delegated to any tenant and thus remain unused. The required flowspace regarding tenant K and switch A must be defined according to the aforementioned tuple regarding switch-wide slicing, as the 4 following rules (1 rule per port); (i) tenant K, in/out port 1, switch A, VLAN i, (ii) tenant K, in/out port 3, switch A, VLAN i, (iii) tenant K, in/out port 4, switch A, VLAN i, and (iv) tenant K, in/out port 5, switch A, VLAN i.

Similarly, in port-wide slicing method, non-overlapping flowspace rules must be created regarding each topology switch port. As shown in Figure 3(b), tenant K has been assigned ports 1 and 5 of switch A while tenant L ports 2 and 3 of the same switch. Both tenants use the same logical separator (i.e., VLAN i). However, if tenant K selects port 2 or 3 as the egress port, an isolation policy violation would occur. To avoid this conflict, non-overlapping flowspace rules for tenants K and L including switch ports must be specified.

IV. Experimental Evaluation

A. Evaluation Methodology

In order to quantify control-plane slicing performance and possible tradeoffs among slicing methods, we implemented an instance of an FSP rule engine [27]. The FSP engine constitutes a mechanism translating infrastructure management-plane slicing policies into VN mapping control-plane rules and supports the proposed slicing methods. This engine takes as inputs tenant requests for virtual networks and substrate topologies and creates the flowspace rules that are required for the control-plane slicing among tenants.

In our experiments, for all the scenarios, we generated flowspace rules using the FSP engine. These rules were injected into the FlowVisor to assess its time overhead and

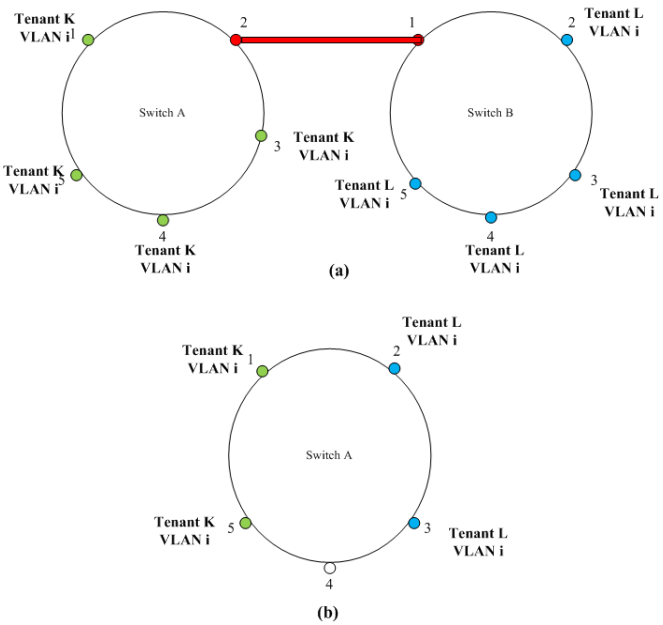


Figure 3. Isolation enforcement, using the (a) Switch-wide slicing, and (b) Port-wide slicing methods

memory consumption. The network topologies used in our experiments included Internet2/OS3E [7] and GÉANT [8], with geographical coordinates of nodes and distances among them imported using data from [28]. Tenant requests for end-to-end shortest paths were computed via shortest path algorithms based on propagation delays.

Tenants had the choice of specifying the virtual network logical separator value, referred as bound requests, or leave the selection to the management plane via unbound requests. The output of the FSP engine (table of flowspace rules) was generated in accordance with the flowspace isolation slicing policy described in Section III.

In our experimental setup, each tenant request was associated with a randomly generated virtual topology, belonging to one out of three categories: star, simple point-to-point paths and disjoint paths. The choice of these particular virtual topologies was based on the fact that major WAN infrastructure tenants are either (i) content providers (e.g. Netflix, Hulu) requesting hub-and-spoke Content Delivery Network (CDN) topologies, or (ii) emerging alternative telecom providers requiring point-to-point or disjoint paths services to configure virtual overlays (e.g., towards provision of Virtual Network Operator services). Moreover, by combining simple point-to-point paths and star topologies, complex topologies can be created (e.g., various hub-and-spoke topologies). The amount of disjoint paths was dictated by the maximum number of the existing disjoint paths per substrate topology (algorithm in [29] was used for disjoint path finding).

For our experiments (involving from 1,000 to 16,000 VN requests), we generated the following mix scenarios:

- **Mix1:** 49% of the requests were for star topologies and 49% for simple paths, while the remaining 2% for disjoint paths. All of them were bound requests.

- **Mix2:** 69% of the requests were for star topologies and 29% for simple paths, while the remaining 2% for disjoint paths. All of them were bound requests.
- **Mix3:** 69% of the requests were for star topologies and 29% for simple paths, while the remaining 2% for disjoint paths. 20% of the requests per category were unbound.
- **Mix4:** 69% of the requests were star topologies and 29% simple paths, while the remaining 2% for disjoint paths. All of them were unbound requests.

B. *Evaluation Results*

1) Acceptance Ratio

In Figure 4, we present the resulting acceptance ratio, when our service runs on top of Internet2/OS3E and GÉANT for 4,000 and 16,000 randomly generated virtual network requests using the aforementioned mixtures (i.e., mix {1-4}).

In Figure 4, it is demonstrated that for bound requests (mix1, mix2), the port-wide slicing method scales better for large amounts of requests (e.g. 16,000) with accepted requests 6 out of 10. In scenarios involving a small percentage of unbound requests (mix3), the port-wide slicing method accepts about 8 out of 10, for a total of 16,000 requests. For exclusively unbound requests (mix4), the port-wide method provides almost perfect utilization of resources.

In Figure 5 and Figure 6, we report the computed acceptance ratio, when the FSP engine runs on top of diverse real network topologies for 1,000, 4,000, 8,000 and 16,000 randomly generated virtual network requests. Small-sized topologies consisted of 8 to 24 nodes, medium sized topologies of 25 to 44 nodes and large sized topologies of more than 45 nodes. Figure 5 shows the acceptance ratio in case of mix2 and Figure 6 in case of mix3. We report results only for mix2 and mix3 since they represent the most intriguing usage scenarios. It is shown that port-wide and switch-wide slicing methods have a superior performance as the number of requests increase compared to domain-wide slicing. The port-wide slicing method performs better compared to the others, especially in large sized topologies.

2) Flowspace rule tables

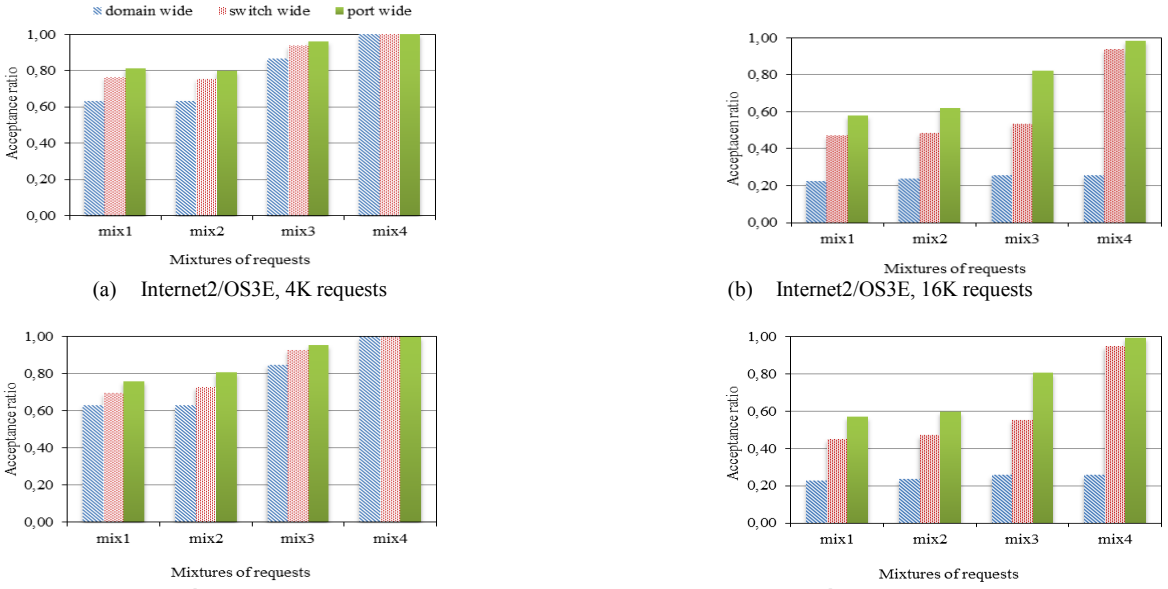
As described above, the switch and the port-wide slicing methods provide high performance in terms of acceptance ratio, especially in cases of large amount of tenant requests. However, this is accomplished at the cost of increasing the number of flowspace rules to be established during the control-plane slicing process.

In Figure 7, we present the number of required rules for switch-wide and port-wide cases, normalized to the number of rules required for the domain-wide slicing. These comparisons were performed for 4,000 tenant requests over substrate topologies consisting of 6 up to 81 infrastructure nodes. The tables of flowspace rules were generated according to the policy described in Section III. The results refer to mix2 and mix4 for switch and port-wide slicing. The first (mix2) was found to represent the worst case scenario for exclusively bound requests, while the second (mix4) involves the largest number of accepted requests, thus generating the largest number of rules.

Moreover, in Figure 7, we plot the linear regression lines of our measurements. Note that the slopes of lines referring to the port-wide slicing method are greater than their switch-wide slicing method equivalent: in domain-wide slicing, a tenant request is translated into a single flowspace rule, whereas, in switch and port-wide slicing, into multiple flowspace rules.

3) Proxy Controller performance overhead

A Proxy Controller adds performance overhead to multi-tenant SDNs, when control actions require message crossing between the control and data-plane layers (e.g. flow establishment/deletion to OF switches).



(a) Internet2/OS3E, 4K requests

(b) Internet2/OS3E, 16K requests

(c) GÉANT, 4K requests

(d) GÉANT, 16K requests

Figure 4. Acceptance ratio

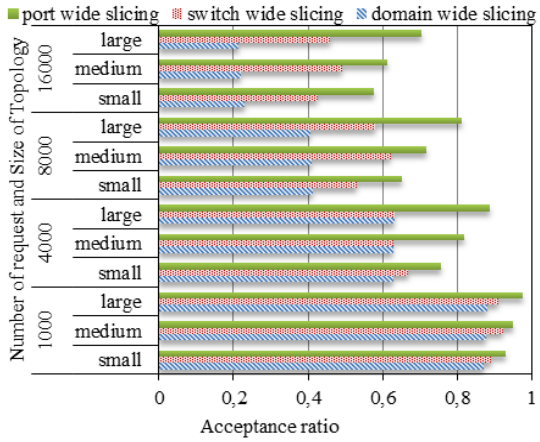


Figure 5. Acceptance ratio in small, medium and large topologies for mix2

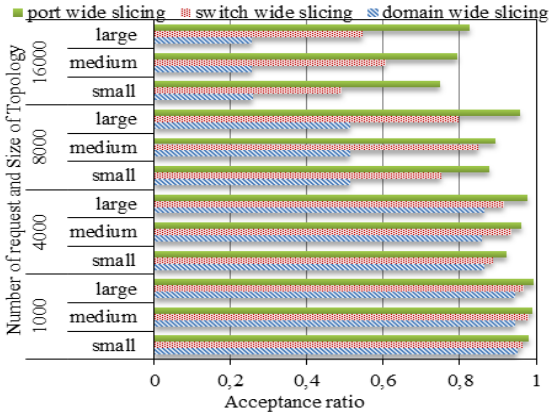


Figure 6. Acceptance ratio in small, medium and large topologies for mix3

We report measurements obtained by using FlowVisor version 1.4 as a proxy controller. This version exhibits a greatly improved flowspace lookup performance due to advanced hashing algorithms, described in [30], compared to the original linear flowspace lookup process [9]. The flowspace rules generated by the FSP engine were injected into the FlowVisor and measurements were obtained for the GÉANT backbone topology.

For the measurement of time overhead, we used the method described in [9]. This involves measuring the time between receiving a control packet from an OpenFlow switch (southbound interface of Proxy Controller) and sending this packet to the tenant OF Controller (northbound interface) using libpcap [31]. The corresponding results are presented in Table I. These results were obtained using the mix2 of tenant requests and the port-wide slicing method.

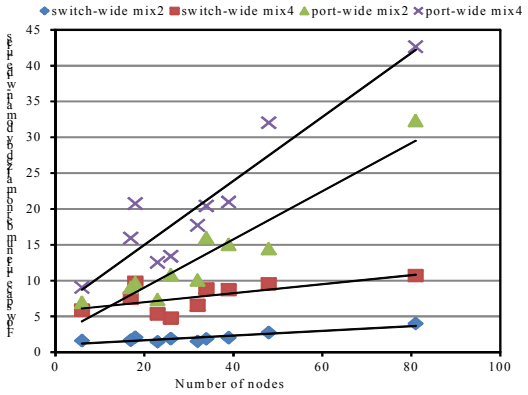


Figure 7. Flowspace rule number for switch-wide and port-wide methods, normalized by domain-wide rules, for mix2 and mix4 (4,000 requests)

The FlowVisor, as shown in Table I, adds a small time overhead to the system, even in the case of large number of rules (i.e., 180,000). Note that, the instantiation of the required rules resulted in large memory consumption by the Proxy Controller, a fact that is not considered a severe limitation in modern generic-purpose servers. Thus, a Proxy Controller can, in principle, efficiently handle the flowspace rules generated by the FSP engine. Finally, we observed in our experiments that the rule insertion time is minor and independent of the current flowspace table size. We postulate, therefore, that our engine is quite robust in injecting flowspace rules and scales smoothly with increasing rates of tenant requests.

TABLE I. FLOWVISOR TIME OVERHEAD AND MEMORY CONSUMPTION FOR GEANT BACKBONE TOPOLOGY

Tenant requests	**GÉANT backbone topology**		
	Generated rules	***Time overhead (ms)***	***Memory (MB)***
1K	17K	4.7	672
2K	45K	5	1852
4K	100K	5.4	4050
6K	150K	5.5	6102
7K	180K	5.9	7500

V. CONCLUSIONS AND FUTURE WORK

In this paper, we studied two prevailing architectural approaches for multi-tenant OpenFlow-enabled Software Defined Networks that can rely on a flowspace slicing policy in order to enforce isolation among tenants. To that end, we proposed three network control-plane slicing methods: (i) domain-wide slicing, (ii) switch-wide slicing and (iii) port-wide slicing. Our experimental measurements showed that the domain-wide slicing requires the smallest number of flowspace rules, while the port-wide is the most demanding.

Furthermore, we introduced and prototyped a Flowspace Slicing Policy (FSP) rule engine, a mechanism that translates infrastructure management-plane slicing policies into VN mapping control-plane rules. The FSP engine enforces slice isolation within a multi-tenant SDN, independent of the virtual network mapping algorithm. We demonstrated that the port-wide slicing is more efficient in terms of tenant request acceptance ratio for a variety of scenarios involving a large number of virtual network requests on top of multiple WAN topologies. Our experimental feasibility evaluation showed that such flowspace slicing policies can be efficiently injected into a typical Proxy Controller with respect to time overhead and memory consumption, even for the demanding port-wide slicing method.

As future work, we plan to extend the FSP engine in order to support slicing methods compliant with the concept of the Centralized Computation Model of MPLS Path Computation Element (PCE) [32]. PCE functionalities would be performed within our FSP engine and the corresponding Path Computation Clients (PCCs) would be responsible for the VN tenant requests.

REFERENCES

[1] "VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks," RFC 7348 (Informational), http://datatracker.ietf.org/doc/draft-mahalingam-dutt-dcops-vxlan

[2] "NVGRE: Network Virtualization using Generic Routing Encapsulation," (Internet Draft), http://datatracker.ietf.org/doc/draft-sridharan-virtualization-nvgre

[3] "Open Networking Foundation, Software-Defined Networking: The New Norm for Networks," ONF White Paper. https://www.opennetworking.org/

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Computer Communication Review, vol. 32, no. 2, pp. 69–74, 2008.

[5] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, et al. "Onix: A Distributed Control Platform for Large-scale Production Networks," in Proceedings of USENIX OSDI, page 351-364, 2010.

[6] "NTT Communications' Enterprise Cloud Goes Global," http://www.ntt.com/aboutus_e/news/data/20130220.html

[7] "Internet2 Open Science, Scholarship and Services Exchange," http://inddi.wikispaces.com/Internet2-based+NDDI

[8] "GÉANT Topology 2012," http://www.geant.net/Resources/Media_Library/Pages/Maps.aspx

[9] R. Sherwood, G. Gibb, K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, "Can the production network be the testbed?" in Proceedings of USENIX OSDI, Vancouver, Canada, 2010.

[10] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 17–29, 2008.

[11] "Flowspace Firewall," http://globalnoc.iu.edu/software/sdn.html

[12] M. Chowdhury, M. R. Rahman, R. Boutaba, "ViNEYard: virtual network embedding algorithms with coordinated node and link mapping," IEEE/ACM Transactions on Networking, vol. 20, Issue 1, pp. 206-219, 2012.

[13] C. Papagianni, A. Leivadeas, S. Papavassiliou, V. Maglaris, C. Cervello-Pastor, A. Monje, "On the optimal allocation of virtual resources in cloud computing networks," IEEE Transactions on Computers, vol. 62, no. 6, pp. 1060–1071, 2013.

[14] M. Casado, T. Koponen, R. Ramanathan, S. Shenker, "Virtualizing the network forwarding plane," in Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow (PRESTO '10), ACM , New York, USA, Article 8, 2010.

[15] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, G. Parulkar, "OpenVirteX: A Network Hypervisor," Open Networking Summit, 2014.

[16] R. Doriguzzi Corin, M. Gerola, R. Riggio, F. DePellegrini, E. Salvadori, "VeRTIGO: Network Virtualization and Beyond," Software Defined Networking (EWSDN), 2012 European Workshop on SDN, vol., no., pp.24, 29, 2012.

[17] R. Riggio, F. De Pellegrini, E. Salvadori, M. Gerola, C. R. Doriguzzi Corin, "Progressive virtual topology embedding in OpenFlow networks," Integrated Network Management (IM 2013), IFIP/IEEE International Symposium on Integrated Network Management, Ghent, vol., no., pp.1122,1128, 2013.

[18] D. Drutskoy, E. Keller, J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," Internet Computing, IEEE, vol.17, no.2, pp.20,27, 2013.

[19] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, S. Shenker, "NOX: towards an operating system for networks," SIGCOMM Computer Communication Review 38, 3, 105-110, 2008.

[20] B. Heller, R. Sherwood, N. McKeown, "The controller placement problem," in Proceedings of the 1st workshop on Hot topics in Software Defined Networks (HotSDN '12). ACM, New York, USA, pages 7-12, 2012.

[21] "Terminology for Policy-Based Management," RFC 3198 (Informational), http://datatracker.ietf.org/doc/rfc3198

[22] "Policy Core Information Model (PCIM) Extensions," RFC 3460 (Proposed Standard), http://datatracker.ietf.org/doc/rfc3460

[23] "DMTF, Distributed Management Task Force," http:// www.dmtf.org

[24] J. Strassner, "Policy-based Network Management: Solutions for the Next Generation," M. Kaufmann, ISBN-13: 978-1558608597, 2003.

[25] A. Farrel et al, "Network Management Know It All," M. Kaufmann, ISBN-13: 978-0123745989, 2011.

[26] "OpenFlow Protocol version 1.4.0 specification" https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.4.0.pdf

[27] "FlowSpace Policy Rule Engine", [online], https://github.com/spirosmastorakis/FSP_Engine

[28] "The Internet Topology Zoo project," http://www.topology-zoo.org

[29] R. Bhandari, "Optimal physical diversity algorithms and survivable networks," Proc. Second IEEE Symposium on Computers and Communications 1997, Alexandria, Egypt, pp. 433-441, 1997.

[30] D. Knuth, "The Art of Computer Programming, Volume 3: Sorting and Searching," Second Edition, Chapter 6.3, page 492. Addison Wesley, 1997.

[31] tcpdump/libpcap, http://www.tcpdump.org

[32] "A Path Computation Element (PCE)-Based Architecture" (Informational), https://datatracker.ietf.org/doc/rfc4655