AutoSlice: Automated and Scalable Slicing for Software-Defined Networks

Zdravko Bozakov and Panagiotis Papadimitriou
Institute of Communications Technology, Leibniz University of Hannover, Germany
{zdravko.bozakov, panagiotis.papadimitriou}@ikt.uni-hannover.de

ABSTRACT

We present AutoSlice, a virtualization layer that automates the deployment and operation of software-defined network (SDN) slices on top of shared network infrastructures. AutoSlice enables substrate providers to resell their SDN to multiple tenants while minimizing operator intervention. At the same time, tenants are given the means to lease programmable network slices enabling the deployment of arbitrary services based on SDN principles. We outline the control plane architecture of AutoSlice and discuss the most challenging aspects of the forwarding plane design with emphasis on scalability.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

Software-Defined Network, Network Virtualization, Caching

1. INTRODUCTION

Network virtualization comprises a viable solution for the concurrent deployment and operation of isolated network slices on top of shared network infrastructures [1]. The emerging SDN paradigm facilitates the deployment of network services, by combining programmable switching hardware, such as OpenFlow, centralized control and network-wide visibility. These salient properties of SDNs can enable network tenants to take control of their slices, implementing custom forwarding decisions, security policies and configuring access control as needed.

A fundamental building block for SDN virtualization is FlowVisor [2], which enables slicing of the flow table in OpenFlow switches by partitioning it into so-called flowspaces. As a result, switches can be manipulated concurrently by multiple controllers. Nevertheless, the instantiation of an entire vSDN topology is non-trivial, as it involves numerous operations, such as mapping virtual SDN (vSDN) topologies, installing auxiliary flow entries for tunneling and enforcing flow table isolation. Since these operations require considerable planning and management resources, we aim to develop a transparent virtualization layer, or SDN hypervisor, which automates the deployment and operation of arbitrary vSDN topologies with minimal intervention by the substrate operator. In contrast to previous SDN virtualization efforts [3, 4], we focus on the scalability aspects of the hypervisor design. Furthermore, AutoSlice optimizes resource utilization and mitigates flow-table limitations by monitoring flow-level traffic statistics. In the remainder of this paper, we discuss the control and forwarding plane architecture of AutoSlice.

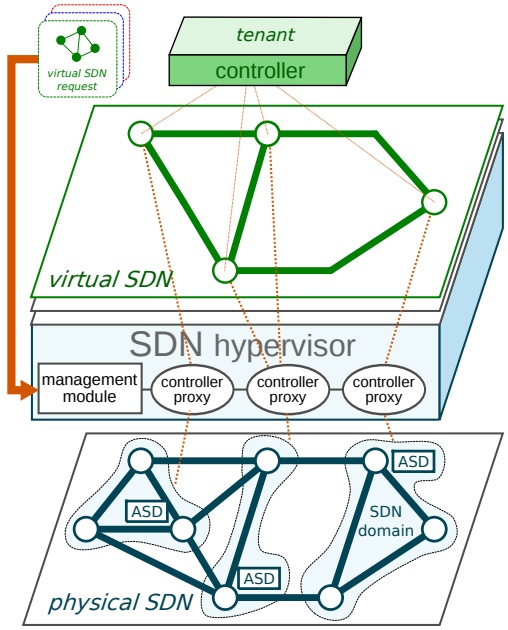

Figure 1: AutoSlice architecture overview.

2. CONTROL PLANE OVERVIEW

We consider a network infrastructure provider offering vSDN topologies to multiple tenants. A tenant's vSDN request comprises a set of nodes and links with certain requirements, such as switching capacity, link bandwidth, and location. Once a vSDN has been deployed, the tenant's controller can install flow entries as if accessing a dedicated physical SDN. We make no assumptions about the type of

CoNEXT Student'12, December 10, 2012, Nice, France.

service, i.e., the packet forwarding and processing rules that the tenant would like to deploy. We only assume that the physical substrate consists of OpenFlow switches, each containing a flow table which can be sliced into logical segments.

We propose a distributed hypervisor architecture, depicted in Fig. 1, which can handle large numbers of flow table control messages from multiple tenants. The hypervisor comprises a *management module* (MM), and multiple *controller proxies* (CPX) which are used to evenly distribute the control load. We segment the physical substrate into multiple *SDN domains* and assign a dedicated CPX to each domain. Each CPX transparently manages the access to the corresponding domain switches. Currently, we assume that all SDN domains are operated by a single SDN provider; however, multi-provider SDN slicing is possible by leveraging on network virtualization architectures, such as [1].

Upon receiving a request, the MM maps the virtual SDN topology to the resources available in each SDN domain, and assigns a subset of logical resources to each CPX. Subsequently, every CPX instantiates the allocated topology segment by installing *infrastructure flow entries* in its domain, which unambiguously bind traffic to a specific logical context using tagging. Since isolation between tenants is essential, each CPX performs policy control on the flow table accesses and ensures that the resulting flow entries are mapped onto non-overlapping flowspaces. All control communication between a tenant's controller and the forwarding plane is redirected through the CPX responsible for the corresponding switch. Before installing a tenant flow entry to a switch, the proxy rewrites the control message, such that all references to virtual resources are replaced by the corresponding physical entities, and appropriate traffic tagging actions are appended. The state of each virtual node in a given SDN domain is maintained solely by the corresponding proxy. Consequently, each CPX can independently migrate virtual resources (e.g., nodes, links) within its domain in order to optimize intra-domain resource utilization. Global optimizations are coordinated by the MM.

The transparent control message translation enables tenants to install arbitrary packet processing rules within an assigned SDN slice, without adversely affecting concurrent users. At the same time, the automation of the infrastructure setup minimizes SDN operator intervention.

3. FORWARDING PLANE

We now consider the main challenges of the forwarding plane design. In a multi-tenant environment a large number of logical flow tables must be mapped onto the memory of a single substrate switch. The CPX ensures the isolation of all virtual flow tables and also guarantees that all packet processing actions are applied in the correct sequence in case a connected group of virtual nodes is mapped to the same switch (e.g., using a loopback interface). The scalability of the platform would be severely restricted by the limited flow table size in OpenFlow switches, which is typically in the order of several thousands of entries. To overcome this limitation, we deploy so-called *auxiliary software datapaths* (ASD) in the substrate network, similarly to [5]. Each SDN domain is assigned an ASD consisting of a software switch, (e.g., OpenVSwitch), running on a commodity server. In contrast to an OpenFlow switch, the available main memory in a server is sufficient for storing a full copy of all logical flow tables required by the corresponding ASD. However, despite the recent advances in software-based datapath architectures and commodity servers, the divide between commodity and specialized hardware still remains, with the latter offering at least an order of magnitude larger switching capacity.

To circumvent these limitations, we exploit the Zipf property of aggregate traffic, i.e., the fact that a small fraction of flows account for most of the traffic volume. We use ASDs for handling low-volume traffic flows (mice), while caching a small number of high-volume (elephants) flows in the dedicated switches. To this end, a set of low-priority, infrastructure entries route traffic from the domain edge to the ASD, when no high-priority flow entry has been cached at a domain switch (i.e., acting as a fallback path). The CPX selects the flow entries that will be cached, so that most of the traffic is offloaded from the ASD. In addition, CPX ensures that cached flow entries do not alter the semantic integrity of the flow table rules by re-encoding flow entries as needed. An effective traffic offloading approach for IP forwarding has been evaluated in [6]. The varying traffic demands and diverse forwarding rules deployed by tenants in their vSDNs introduce additional complexity on flow caching decisions. To this end, we are developing a caching mechanism which exploits the flow-level statistics exposed by OpenFlow switches. In particular, we make offloading decisions based on inferred traffic characteristics, such as traffic burstiness.

4. IMPLEMENTATION AND FUTURE WORK

We have implemented a small-scale prototype, as proof-of-concept of the AutoSlice architecture, using Pronto 3295 switches and commodity servers with OpenVSwitch as ASDs. We employ OpenFlow 1.0 and VLAN for packet tagging. Our prototype implementation currently comprises modules for switch flow table access control (built on top of FlowVisor), control traffic classification, flow caching, and vSDN mapping. Our flow caching approach relies on random sampling which ensures undistorted estimates of traffic characteristics while mitigating the control load associated with flow statistics requests. In addition, we aim to study the dynamic segmentation of the substrate into SDN domains as vSDNs are provisioned or released, the suitability of existing virtual network embedding algorithms for vSDN mapping, the collaboration among CPX for operations across multiple SDN domains (e.g., vSDN migration) and the efficiency of flow caching techniques.

5. REFERENCES

[1] G. Schaffrath, et al., Network Virtualization Architecture: Proposal and Initial Prototype, Proc. ACM SIGCOMM VISA 2009.

[2] R. Sherwood, et al., Can the Production Network Be the Testbed?, Proc. USENIX OSDI 2010.

[3] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, Virtualizing the network forwarding plane. Proc. ACM CONEXT PRESTO 2010.

[4] E. Salvadori, et al., Generalizing virtual network topologies in OpenFlow-based networks, Proc. IEEE GLOBECOM 2011.

[5] M. Yu, et al., Scalable flow-based networking with DIFANE, Proc. ACM SIGCOMM 2010.

[6] N. Sarrar, et al., Leveraging's Zipf's Law for Traffic Offloading, ACM SIGCOMM CCR, 2012.