Serial Composition of Heterogeneous Control Planes

Kirill Kogan, Advait Dixit, and Patrick Eugster
Purdue University

1 Motivation and Problem Statement

Standardized data-path provision as achieved by OpenFlow enables the *integration of network elements from different vendors* in the same forwarding plane, with these elements being managed by the same logically centralized control plane. However, there is *no* clear abstraction that enables the *integration of control planes from different vendors* into a single virtually centralized controller. Yet, today's software-defined network (SDN) control planes are highly homogeneous and require networks (or slices thereof) to be controlled by the same controller platform [6, 2, 4, 1, 9]. Hence to deploy a new service in a network, an operator needs to acquire and integrate new implementation modules into the existing controller platform. Though some earlier work has considered increasing the flexibility of controllers by allowing dynamic addition and removal of services, all integrated services need to obey specific constraints of the integrating platform [3, 11].

There is a very stong incentive to integrate heterogeneous control planes. Modern networks are intelligent, and require implementation of sophisticated services such as advanced VPN services, deep packet inspection, firewalling, intrusion detection – to list just a few. Moreover, this list continues to grow, increasing the need for methods to implement new network policies. However, not all services may be available on the same controller platform. Furthermore, it is unlikely that a single controller vendor will have the best-in-class implementation for all services. As a consequence, network operators are brought to choose between not deploying a service at all or moving to a different controller platform, which is expensive, disruptive and often simply infeasible. Thus, network operators are in need of constructors for flexible implementation of policies (that consist of services from various controller vendors, ideally, transparently to the services of integrated controllers) which allow for flexible and sound assemblage.

A standardization of datapath provision makes an integration of services from heterogeneous vendors possible. Previous research has tackled two types of service integration: *parallel* and *serial*. Parallel integration allows a network operator to slice the managed network and assign a controller to each slice; it is important to note that in this case each slice is an independent network, and services integrated from different controllers are not applied to the same traffic. Flowvisor [13] and Frenetic [5] are two implementations of parallel service integration. In the serial case the composed policy is an ordered set of services that can be applied to the same traffic (in the same network element). Pyretic [12] is an example of such serial service composition. While the problem of parallel integration of heterogeneous controllers is already resolved by Flowvisor [13], the serial integration of services from different controllers is still not addressed since Pyretic [12] assumes that all serially composed services are running on top of the same controller platform (or requires a common northbound API for the composed controllers). In this work we are going beyond the previous work and propose an abstraction for serial service integration, transparently to the composed controllers. A key trait of our proposed abstraction is that it does not require any additional standardization.

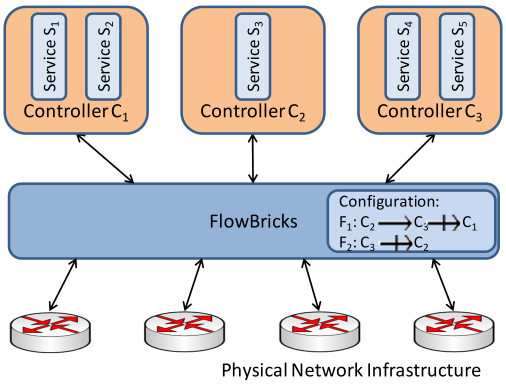

Figure 1: FlowBricks system architecture.

2 Design of FlowBricks

To address these challenges, we present FlowBricks, a flexible SDN architecture for serial composition of services from heterogeneous controllers. FlowBricks is a virtual layer between the heterogenous control and forwarding planes as shown in Figure 1. In a nutshell, FlowBricks acts as a proxy between a forwarding plane

and heterogeneous controllers that run desired subsets of features. From the perspective of the forwarding plane, FlowBricks is the SDN controller, while switches use the standard datapath protocol to communicate with it. From the perspective of controllers, FlowBricks is the forwarding plane. It initiates connections (one for each switch) with each controller and communicates using the standard datapath protocol.

As shown in Figure 1, the system has a two-level hierarchical configuration. Each controller is configured to provide a subset of the desired services and FlowBricks is configured to combine services from different controllers in the desired sequence. To avoid additional standardization and allow any order of serial composition of services, we require that the set of services at each controller be independent of those running on other controllers. For instance, the configuration: $F_1 : C_2 \rightarrow C_3 \mapsto C_1$ specifies that services should be applied in the sequence S_3, S_4, S_5, S_1, S_2 to flow F_1. "$\mapsto$" indicates a fence. When packet processing reaches a fence, actions from previous services should be applied to the packet before proceeding to the next service. "$\rightarrow$" indicates that the next service should see the unmodified packet. The actions from all services are applied to the packet when packet processing reaches a fence, or at the end of pipeline execution.

Each controller sends messages to FlowBricks to configure flow tables according to the services that it implements. FlowBricks modifies these messages to install a combined packet processing pipeline on the switch. FlowBricks achieves this by installing flow tables from all controllers. It then inserts flow table entries to direct execution between flow tables of different controllers according to the sequence specified in its configuration.

In designing, prototyping, and evaluating FlowBricks, we make the following contributions:

- We built a prototype of FlowBricks using Floodlight [4] and composed policies from services implemented on different controllers. Using our prototype, we conducted experiments to show that FlowBricks does not significantly impact response time and scalability of the control plane.
- Floodlight communicates with switches over OpenFlow [10]. For most parts, implementation of FlowBricks in OpenFlow is feasible. However, we identified four minor changes to OpenFlow required to support FlowBricks.

In the future we plan to employ techniques that address the fundamental space-performance tradeoff during integration of controllers similarly to [7, 8].

References

[1] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A system for scalable OpenFlow control," Tech. Rep. TR10-11, CS Department, Rice University, Dec. 2010.

[2] A. Dixit, F. Hao, S. Mukherjee, T.V. Lakshman, and R. Kompella, "Towards an Elastic Distributed SDN Controller," in *HotSDN*, 2013.

[3] D. Erickson, "The Beacon OpenFlow Controller," in *HotSDN*, 2013.

[4] "Floodlight," `http://floodlight.openflowhub.org`.

[5] N. Foster, H. Harrison, M. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A Network Programming Language," in *ICFP*, 2011.

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. Mckeown, and S. Shenker, "NOX: Towards an Operating System for Networks," in *SIGCOMM CCR*, 2008.

[7] Alexander Kesselman, Kirill Kogan, et al., "Space and speed tradeoffs in TCAM hierarchical packet classification," *J. Comput. Syst. Sci.*, 2013.

[8] Kirill Kogan, Sergey I. Nikolenko, William Culhane, Patrick Eugster, and Eddie Ruan, "Towards efficient implementation of packet classifiers in sdn/openflow," in *HotSDN*, 2013, pp. 153–154.

[9] T. Koponen et al., "Onix: A Distributed Control Platform for Large-scale Production Networks," in *OSDI*, 2010.

[10] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, et al., "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM CCR*, 2008.

[11] M. Monaco, O. Michel, and E. Keller, "Applying Operating System Principles to SDN Controller Design," in *HotNets*, 2013.

[12] C. Monsanto, J. Reicha, N. Foster, J. Rexford, and D. Walker, "Composing Software-Defined Networks," in *NSDI*, 2013.

[13] R. Sherwood, G. Gibb, K. K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Can the Production Network Be the Testbed?," in *OSDI*, 2010.