A Slick Control Plane for Network Middleboxes

Bilal Anwer
Georgia Tech
bilal@gatech.edu

Theophilus Benson
Duke University
tbenson@cs.duke.edu

Nick Feamster
Georgia Tech
feamster@cc.gatech.edu

Dave Levin
University of Maryland
dml@cs.umd.edu

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]; C.2.1 [**Network Architecture and Design**]: Centralized Networks; C.2.3 [**Network Operations**]: Network Management

Keywords

Middlebox, Network Management, Software-Defined Networking

1. INTRODUCTION

Recent advances in software-defined networking (SDN), particularly OpenFlow [5], have made it possible to implement and deploy sophisticated network policies with relatively simple programs. The simplicity arises in large part due to a simple "match/action" interface that OpenFlow provides, by which a programmer can specify actions to take on packets that match particular characteristics (e.g., "forward all port-53 traffic on a faster path").

To date, however, the space of such policies that can be easily implemented in an SDN centers on the control plane—while OpenFlow provides a rich control plane API, it permits very narrow control on the data plane. Expanding the match/action interface could make it possible for network operators to implement more sophisticated policies, e.g., that perform deep packet inspection and operate at the application layer. Yet, expanding OpenFlow's specification is an arduous process, requiring standardization and hardware support—going down that path would, we believe, ultimately result in vertically integrated hardware, the very fate that OpenFlow was arguably designed to avoid.

On the other end of the spectrum we have middleboxes: computing devices that sit on traffic flows' paths, and that have no inherent restrictions on what they can process or store. Middleboxes have historically been vertically integrated, thus, although middlebox manufacturers can create a wide range of data processing devices, network operators remain faced with several key challenges:

- **Placement**: Because the code and hardware are tightly coupled, operators must determine where to place middleboxes *a priori*, potentially resulting in unnecessarily large path inflation. Ideally, the right functionality would always be on the shortest or least congested path.
- **Scaling**: Operators must also anticipate the demands of *each* of their middleboxes, so as to sufficiently provision. This typically results in excessive over-provisioning. Ideally, the functionality would scale up to meet demands, and scale down to conserve resources.
- **Composition**: A given policy might span multiple middleboxes (e.g., the policy might express that an IDS should inform a firewall), but, much like the motivation behind OpenFlow, vertical integration makes it challenging for any given network operator to expand functionality to meet their specific needs.

Broadly speaking, vertically integrated middleboxes make for a network that is rigid (it cannot dynamically reconfigure to meet demands) and inefficient (it requires either extensive over-provisioning or potentially high path inflation).

Recent, promising approaches demonstrate the flexibility that arises from incorporating programmable middleboxes into the network [1–3, 6]. We are inspired by these systems; they require no modifications to OpenFlow, and have been shown to permit implementation of a wide range of middlebox functionality. However, systems that expand the flexibility of middleboxes have, to date, come at the cost of efficiency. CoMb [6], for instance, bundles multiple middlebox functions into a single "hyperapp," making it difficult or impossible to scale out one function without having to scale out the others. Embrane [1], on the other hand, offers extensive support for scaling, but makes it more difficult to compose multiple middlebox functions into a single, more powerful application.

We propose a *control plane for network middleboxes*. Similar to SDN's control plane, we argue for decoupling processing between in-network devices ("middleboxes") and a controller that coordinates among them. Unique to the data plane, we believe that a control plane for network middleboxes should support heterogeneous devices (e.g., NetFPGAs, GPUs, and NPs), so that operators can make use of the most efficient hardware for the appropriate tasks. Finally, and perhaps most crucially, to support dynamic reconfiguration of what functions are running and where in the network they reside, we argue that a middlebox control plane should automate traffic steering and function placement (and migration).

We have designed an implemented *Slick*, a prototypical control plane for network middleboxes. We describe Slick at a high level in the following section.

HotSDN'13, August 16, 2013, Hong Kong, China.
ACM 978-1-4503-2178-5/13/08.

2. SLICK ARCHITECTURE

Slick is designed to support networks that contain a wide range of computing devices—such as standard x86 servers, NetFPGAs, GPUs, or blackbox middleboxes—distributed throughout the network. As a control plane for network middleboxes, Slick's primary goal is to provide the means by which a network operator can easily implement and efficiently deploy sophisticated policies, imposing as little path inflation as possible. Slick decomposes this problem into three main components: (1) a wireline protocol that handles coordination among centralized controllers and distributed middleboxes, (2) a programming model that facilitates composition of middlebox functionality and encourages modularity, and (3) an optimization algorithm run at the controller that adaptively places or migrates middlebox functions and steers traffic. In sum, Slick supports sophisticated policies, and efficiently uses a network's resources without requiring complicated *a priori* provisioning from operators.

2.1 Control Plane Protocol

Slick provides interfaces that allow the controller to install and remove code on computing devices throughout the network. This protocol is similar to OpenFlow's, in that the controller specifies match/action rules, and the middlebox runs the piece of code that matches according to the rules. Slick supports multiple concurrent pieces of code running on a middlebox. A *shim* layer supports this by serving as a form of hypervisor: it multiplexes and demultiplexes messages from the controller (or packets from the network) to the matching pieces of code.

The Slick protocol allows middleboxes to raise asynchronous events to the controller. However, a key addition in the Slick control protocol is that these events can be arbitrary in Slick; programs running on middleboxes can raise custom *triggers*, which serve as an important building block for dynamically composing multiple pieces of code into a single, more powerful application.

2.2 Programming Model

Through the use of triggers, Slick's wireline protocol allows a single policy to be split into separate executables that run across multiple middleboxes concurrently. We believe this model to be both important in achieving efficiency, as it allows us to scale out some parts of a policy (e.g., the firewall) without having to scale out others (e.g., an IDS). Furthermore, and somewhat anecdotally, we have found that it simplifies implementation of new policies and encourages code reuse.

Slick supports this programming model natively, viewing the implementation of a policy as consisting of two broad pieces:

- **Applications** represent policies; they dictate what data plane functionality should be performed and on what traffic. For example, applications can specify to "check all DNS queries against a blacklist." While applications specify *what* processing should occur, they need not specify *where* in the network the processing must occur.
- **Elements** perform the data plane processing similar to what middleboxes do today—e.g., implementing a firewall. Slick elements are intended to be smaller and more modular than traditional, vertically-integrated middleboxes.

At a high level, this decomposition should appear similar to the model of Click modular routers [4]. This is intentional: our informal goal was to make Slick as easy to program as Click. Slick differs, however, in that elements may be distributed across multiple (potentially heterogeneous) middleboxes. Slick meets our informal goal by keeping hidden from the programmer the details of where the code is running at any point in time; this is instead in an automated manner by the Slick controller.

2.3 Slick Controller

Slick distributes elements throughout the network and uses the controller to determine where to install elements and to provide coordination between applications and elements. There are two key features a Slick controller provides:

- **Support for heterogeneity:** Not all elements can run on all network devices; some require dedicated GPUs for parallelism, or FPGAs to support higher demands. The technical challenge from the controller's perspective is to determine on what set of devices a given element can be installed. Slick can support elements that are written to run on specialized hardware, such as cryptographic coprocessors, NetFPGAs, GPUs, or even blackbox middleboxes. We believe this to be an important feature: it allows operators to make use of a wide range of devices (including those they may already have installed), and it allows element programmers to make use of highly efficient hardware, which may be necessary to perform data plane processing at line rates.
- **Element placement and traffic steering**: As a result of Slick's programming model and wireline protocol, elements are the indivisible elements of execution. The Slick controller determines where to place elements, and how to steer the matching traffic to them. Performing both concurrently distinguishes Slick from prior work—to the best of our knowledge, prior approaches assume that at least one is fixed—and allows Slick to make more efficient use of the network's resources. It also raises key technical challenges: in particular, determining placement and path selection in an online manner requires approximation algorithms that reduce path inflation, eliminate network congestion, and make judicious use of element migration.

Interestingly, these features are sufficient to support dynamic scaling of elements to meet varying traffic demands. Note that, when an element passes a high (or low) watermark, it can raise a trigger, informing its application that it should request the controller to install another (or remove an existing) element. The controller's optimization algorithm would then reconsider its element placement and path selection to accommodate the new (or removed) elements.

Acknowledgments

This work is supported in part by DARPA through the U.S. Navy SPAWAR under contract N66001-11-C-4017.

3. REFERENCES

[1] Powering virtual network services. `http://embrane.com`.
[2] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting parallelism to scale software routers. In *SOSP*, 2009.
[3] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *SIGCOMM*, 2008.
[4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.
[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *ACM CCR*, Apr. 2008.
[6] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *NSDI*, 2012.