Extending SDN to Handle Dynamic Middlebox Actions via FlowTags[*]

Seyed Kaveh Fayazbakhsh[†] Luis Chiang[‡] Vyas Sekar[†] Minlan Yu[◇] Jeffrey C. Mogul[⋆]
[†]*Carnegie Mellon University* [‡]*Deutsche Telekom Labs* [◇]*USC* [⋆]*Google*

1 Introduction

Software-defined networking (SDN) seeks to simplify and enhance network management by decoupling the management logic from its implementation. Our overarching vision is to integrate advanced data plane functions or middleboxes (e.g., firewalls, NATs, proxies, intrusion detection and prevention systems, and application-level gateways) into the SDN fold. This integration, however, is challenging on two fronts: (1) it is difficult to ensure that "service-chaining" policies are implemented correctly [4], and (2) middleboxes hinder management functions such as performance debugging [5].

The root cause of this problem is that as packets traverse the network, they are altered by dynamic and opaque middlebox actions; for instance, proxies terminate TCP sessions, while NATs and load balancers rewrite headers. Thus, the promise of SDN to systematically enforce and verify network-wide policies (e.g., [3]) does not directly extend to networks with middlebox functions.

In this work, we take a pragmatic stance that rather than eliminate or completely rearchitect middleboxes, we should attempt to integrate them into the SDN fold as "cleanly" as possible. To this end, we extend the SDN paradigm in the *FlowTags* architecture by identifying flow tracking as the key to policy enforcement in the presence of dynamic traffic transformations. That is, we need to reliably associate additional contextual information with a traffic flow as it traverses the network, even if the packet headers and contents are modified. Because middleboxes are in the best (and possibly the only) position to provide the relevant contextual information, FlowTags uses minimal extensions to existing middleboxes to add the relevant *tags*, carried in packet headers. SDN switches use the tags as part of their flow matching logic for their forwarding operations. Downstream middleboxes use the tags as part of their packet processing workflows; e.g., a firewall located after a NAT can use the tags to identify the true source IPs and apply the correct set of rules.

2 FlowTags Architecture

In this section, we show how FlowTags extends the SDN paradigm. Conceptually, middleboxes add tags to outgoing packets. These tags provide the missing bindings between packets and their origins and the necessary processing context. The tags are used in the data plane configuration of OpenFlow switches and other downstream middleboxes.

The FlowTags approach has three key aspects. First, middleboxes are *generators* of the tags, and the packet-processing actions of a FlowTags-enhanced middlebox entails adding the relevant tags into the packet header. Second, other middleboxes are tag *consumers*, and their processing actions need to decode the tags. Third, SDN-capable switches in the network use the tags as part of their forwarding actions in order to route packets according to their intended policy.

FlowTags requires neither new capabilities from SDN switches, nor any direct interactions between middleboxes and switches. Switches continue to use current switch-centric APIs such as OpenFlow. The only interaction between switches and middleboxes is indirect, via tags embedded inside packet headers. We take this approach for two reasons: (1) to allow switch and middlebox designs and their APIs to innovate independently; and (2) to retain compatibility with existing SDN standards (e.g., OpenFlow). Embedding tags in packet headers avoids the need for each switch and middlebox to communicate with the controller on every packet when making their forwarding and processing decisions. Further, we retain existing configuration interfaces for customizing middlebox actions; e.g., vendor-specific languages or APIs to configure firewall/IDS rules. The advantage of FlowTags is that administrators can configure these rules without worrying about the impact of intermediate middleboxes.

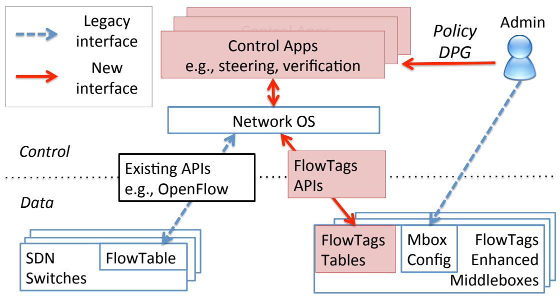

Figure 1: Interfaces between different components in the FlowTags architecture.

As shown in Figure 1, FlowTags extends today's SDN approach along three key dimensions:

1. FlowTags APIs between the controller and FlowTags-enhanced middleboxes to programmatically configure the tag generation and consumption logic.

[*]The full version of this work appears in NSDI 2014 [2].

2. FlowTags controller that configures the tagging-related behavior of the middleboxes and tag-related forwarding actions of SDN switches. The input to the FlowTags controller is the *policy* that the administrator wants to enforce w.r.t. middlebox actions (Figure 1). We extend prior work on static middlebox policy [4] to capture binding of traffic to its origin in the presence of dynamic actions using the notion of a *dynamic policy graph* (or DPG as denoted in Figure 1).

A DPG is a directed graph where each node represents a logical "role" of a type of middlebox function, such as a "firewall." Each logical middlebox node is given a configuration that governs its processing behavior for each traffic class (e.g., firewall rulesets). Administrators specify middlebox configurations in terms of the unmodified traffic entering the DPG without worrying about intermediate transformations. Each edge of the DPG is annotated with the *condition* under which a packet needs to be steered from one middlebox to another. A condition is defined based on (1) the traffic class and (2) the *processing context* of the previous middleboxes (e.g., cache hit/miss in a proxy).

The three main operations of the controller are as follows:

- **Initialization:** Given an input DPG and a physical network topology, we generate a data plane realization of the DPG.
- **Middlebox event handlers:** Corresponding to each physical middlebox instance, the controller maintains two look up tables to handle the middlebox's tag generation and tag consumption queries. We restrict our design of FlowTags to a reactive controller that responds to incoming packets. (Middleboxes cache the controller responses so that they need to query the controller only per new flow.)
- **Switch and flow expiry handlers:** The handlers for *packet-in* messages are similar to traditional OpenFlow handlers; the only exception is that we use the incoming tag to determine the forwarding entry. When a flow expires, we update the look-up tables accordingly so that the corresponding tags can be repurposed.

3. FlowTags-enhanced middleboxes that account for an incoming packet's existing tag when processing the packet and add a new tag based on the context before sending out the processed packet. In our experience, *consumption* inherently precedes *generation* in a given middlebox. The reason is that the packet's current tag can affect the specific middlebox code paths and thus impacts the eventual outgoing tags. We have modified five software middleboxes including Squid (a proxy/cache), Snort (an IDS/IPS), Balance (a TCP-level load balancer), PRADS (a passive monitor), and iptables (configured as a source NAT). Our current approach to extend middleboxes is semi-manual and involved a combination of call graph analysis and traffic injection and logging techniques. In addition to a common library ($\approx$ 250 lines) that implements communication with the controller, adding support for FlowTags required less than 75 lines of custom code.

3 Evaluations

We have implemented the FlowTags controller as a `POX` module and conducted experiments in a virtualized environment (using Mininet and deploying each modified middlebox on a separate KVM instance) to measure the performance and scalability of FlowTags. A synopsis of our evaluations results is as follows:

- In addition to policy enforcement, FlowTags enables new verification and network diagnosis methods that are otherwise hindered due to middlebox actions.
- The run-time, CPU, and memory overhead that supporting FlowTags adds to a middlebox is very low ($<$1%).
- The FlowTags controller run time is linear as a function of topology size with the baseline algorithms but almost constant with a simple optimization that pre-computes and stores network reachability information.
- The overhead of the FlowTags operations over traditional SDN in a FlowTags-enhanced network is negligible. Further, the reduction in TCP throughput a flow experiences in a FlowTags-enhanced network (due to tag consumption/generation queries per flow) compared to a traditional SDN network with middleboxes (but without FlowTags) is $<$4%.
- The number of tag bits we need, based on large real-world traces, with some simple spatial and temporal tag reuse optimizations is not more than 14 bits, which is small enough to be embedded in a variety of packet headers.

In summary, we have shown the feasibility of integrating middleboxes into the SDN fold via systematically tagging traffic flows. FlowTags requires minimal modifications to existing middleboxes that makes it well-suited for vendor adoption. We believe that there are three natural directions for future work: automating DPG generation via model refinement techniques (e.g., [1]); automating middlebox extension using appropriate programming-languages techniques; and, performing holistic testing of the network while accounting for switches and middleboxes.

References

[1] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. CAV*, 2000.

[2] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using FlowTags. In *Proc. NSDI*, 2014.

[3] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: static checking for networks. In *Proc. NSDI*, 2012.

[4] Z. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying Middlebox Policy Enforcement Using SDN. In *Proc. SIGCOMM*, 2013.

[5] W. Wu, G. Wang, A. Akella, and A. Shaikh. Virtual network diagnosis as a service. In *Proc. SoCC*, 2013.