Practical and Incremental Convergence between SDN and Middleboxes

Zafar Qazi[†], Cheng-Chun Tu[†], Rui Miao [⋆], Luis Chiang[†], Vyas Sekar[†], Minlan Yu [⋆]
[†] Stony Brook University [⋆] USC

A Introduction

Managing middleboxes to achieve the performance and security benefits they offer is highly complex. This complexity stems from the need to carefully plan the network topology, manually set up rules to route traffic through the desired sequence of middleboxes, and safeguard for correct operation in the presence of failures/overload. In this context, middleboxes represent an opportunity, a necessity, and a challenge for SDN. They are an opportunity for SDN to demonstrate a practical use-case for L4–L7 functions that the market views as important; they are a necessity given the industry concerns surrounding the ability of SDN to integrate with existing network infrastructure; and they are a challenge as they introduce aspects that fall outside the scope of traditional L2/L3 functions.

The goal of this work is to bring the benefits of SDN for middlebox management *without mandating any placement [6, 8] or implementation [10, 4] constraints on middlebox functions and without changing current SDN standards [5].* We impose these constraints because of two fundamental reasons. First, future middleboxes will be even more diverse and incorporate proprietary logic and hardware components. Second, we want to retain the *minimalism* of existing SDN (e.g., OpenFlow) that has been a key driver for adoption by hardware vendors. While this makes our problem scope more modest relative to the aforementioned proposals, it is arguably more practical and immediately deployable. Addressing the question of incremental convergence between middleboxes and SDN is especially relevant in light of industry concerns surrounding SDN adoption [2, 1, 3].

What makes this problem space interesting is that middleboxes introduce new dimensions for network management that fall outside the purvey of traditional L2/L3 functions. This creates new opportunities as well as challenges for SDN:

- **Composition of middleboxes:** Network policies typically require packets to go through a sequence of middleboxes (e.g., firewall+IDS+proxy). SDN can eliminate the need to manually plan middlebox placements or configure routes to enforce such policies. At the same time, using flow-based forwarding rules that suffice for L2/L3 applications can lead to inefficient use of the available switch TCAM (e.g., we might need several thousands of rules) and also lead to incorrect or ambiguous forwarding decisions (e.g., when multiple middleboxes need to process the same packet).
- **Middlebox load balancing:** Due to the complexity of packet processing (e.g., deep packet inspection), a key factor in middlebox deployments is to balance the processing load to avoid overload [8]. SDN provides the flexibility to implement different load balancing algorithms and avoids the need for operators to manually install traffic splitting rules or use custom load balancing solutions. Unfortunately, the limited TCAM space in SDN switches makes the problem of optimally balancing middlebox load both theoretically and practically intractable.
- **Packet modifications:** Middleboxes modify packet headers (e.g, NATs) and even manipulate session-level behaviors (e.g., WAN optimizers and proxies may use persistent connections). Today, operators have to account for these effects via careful placement or manually reason about the impact of these modifications on routing configurations. By taking a network-wide view, SDN can eliminate errors from this tedious process. Due to the proprietary nature of middleboxes, however, a SDN controller may have limited visibility to set up forwarding rules that account for such transformations.

B Approach Overview

Based on the trajectory of the aforementioned prior work (e.g., [10, 5, 8, 6]), trying to meet the above challenges within the confines of existing SDN interfaces and middlebox implementations seems infeasible at first glance. Perhaps surprisingly, we show that it is possible to address all three challenges using our proposed NIMBLE system.[1] Figure 1 gives an overview of the NIMBLE architecture showing the inputs needed for various components, the interactions between the modules, and the interfaces to the data plane. Building on the SDN philosophy of direct control, we want

[1] NIMBLE = Network-wIde MiddleBox ControLEr

network administrators to only tell NIMBLE *what* processing policy needs to be implemented and not worry about *where* this processing occurs or how the traffic needs to be routed. Drawing on previous middlebox research [10, 9], this is best expressed via a *dataflow* abstraction as shown. Here, the operator specifies different *policy classes* (e.g., external web traffic or internal NFS traffic) and the middlebox processing needed for each class.

The **ResMgr** module takes as input the network's traffic matrix, topology, policy requirements and outputs a set of middlebox processing assignments that implement the policy requirements. This module takes into account both middlebox and switch constraints in order to optimally balance the load across middleboxes. We address the intractability of optimization by decomposing the problem into a hard offline component that accounts for the integer constraints introduced by switch capacities and an efficient online component that balances middlebox load in response to traffic changes.

The **DynHandler** module automatically infers mappings between incoming and outgoing connections of middleboxes that can modify packet/session headers. It receives packets (from previously unseen connections) from switches that are directly attached to the middleboxes. It uses a lightweight *payload similarity* algorithm to correlate the incoming and outgoing connections and provides these mappings to the **RuleGen** module.

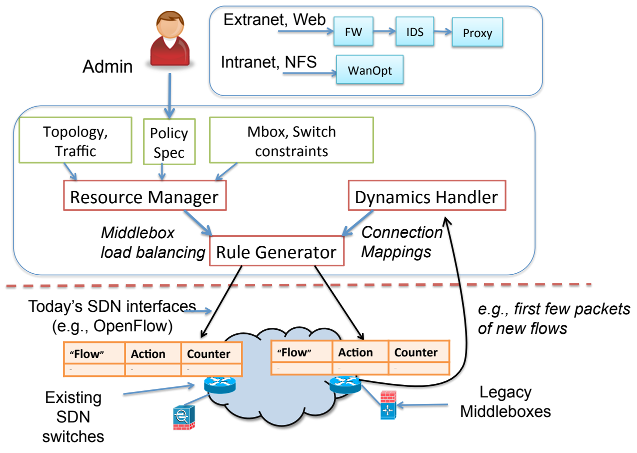

Figure 1: *Overview of the NIMBLE approach for using SDN to manage middlebox deployments*

The **RuleGen** module takes the output of the ResMgr (i.e., the processing responsibilities of different middleboxes) and the connection mappings from the DynHandler and generates data plane configurations to route the traffic through the appropriate sequence of middleboxes to their eventual destination. In addition, the RuleGen also ensures that middleboxes with stateful session semantics receive both the forward and reverse directions of the session. As discussed earlier, we need to make efficient use of the available TCAM space and avoid the ambiguity that arises due to composition. Thus, we provide a compact data plane design that supports these two key properties based on two key ideas: (1) tunnels between switches and (2) using tags to packet headers that annotate each packet with its processing state.

Conceptually, we envision the ResMgr and DynHandler running as controller applications while the RuleGen can be viewed as an extension to the network operating system [7]. In the common case, we envision NIMBLE as a proactive controller that can handle middleboxes that do not modify packet headers. The DynHandler, however, needs to be a reactive component since it needs to infer the connection mappings on the fly.

C Preliminary Evaluation

We have built a proof-of-concept NIMBLE controller using `POX` and leveraging off-the-shelf optimization tools such as `CPLEX`. Using a combination of live experiments on Emulab, large-scale emulations using Mininet, and trace-driven simulations, we show that NIMBLE:

- improves middlebox load balancing 6× compared to today's deployments and achieves near-optimal performance w.r.t new middlebox architectures [10];
- takes only 100ms to bootstrap a network and to respond to network dynamics (e.g., middlebox failure) in a 11-node topology;
- takes less than 1.3 sec to rebalance the middlebox load in the presence of traffic changes and reduces this time 4 orders of magnitude compared to strawman optimization schemes.

References

[1] 2012 Cloud Networking Report. `http://www.webtorials.com/content/2012/11/2012-cloud-networking-report.html`.
[2] ONF Expands Scope; Drives Technical Work Forward. `http://bit.ly/YJeh6a`.
[3] SDN 2013: Market will address Layer 4-7 services. `http://bit.ly/RTcGH1`.
[4] A. Greenlagh et al. Flow Processing and the Rise of Commodity Network Hardware. In *ACM CCR*, 2009.
[5] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella. Toward software-defined middlebox networking. In *Proc. HotNets-XI*, 2012.
[6] G. Gibb, H. Zeng, and N. McKeown. Outsourcing Network Functionality. In *Proc. HotSDN*, 2012.
[7] N. Gude et al. NOX: towards an operating system for networks. In *CCR*, 2008.
[8] J. Sherry et al. Making Middleboxes Someone Else's Problem: Network Processing as a Cloud Service. In *Proc. SIGCOMM*, 2012.
[9] D. A. Joseph, A. Tavakoli, and I. Stoica. A Policy-aware Switching Layer for Data Centers. In *Proc. SIGCOMM*, 2008.
[10] V. Sekar et al. Design and Implementation of a Consolidated Middlebox Architecture. In *Proc. NSDI*, 2012.