Policy Transformation in Software Defined Networks

Nanxi Kang, Joshua Reich, Jennifer Rexford, David Walker
Princeton University
{nkang, jreich, jrex, dpw}@cs.princeton.edu

ABSTRACT

A Software Defined Network (SDN) enforces network-wide policies by installing packet-handling rules across a distributed collection of switches. Today's SDN platforms force programmers to decide how to decompose a high-level policy into the low-level rules in each switch. We argue that future SDN platforms should support automatic transformation of policies by moving, merging, or splitting rules across multiple switches. This would simplify programming by allowing programs written on one abstract switch to run over a more complex network topology, and simplify analysis by consolidating a policy spread over multiple switches into a single list of rules. This poster presents our ongoing work on a sound and complete set of axioms for policy transformation, to enable rewriting of rules across multiple switches while preserving the forwarding policy. These axioms are invaluable for creating and analyzing algorithms for optimizing the rewriting of rules.

Categories and Subject Descriptors

C.2.m [**Computer-Communication Networks**]: Miscellaneous

Keywords

software defined networks, network virtualization, OpenFlow

1. INTRODUCTION

The Software Defined Network (SDN) paradigm enables flexible programming of packet-switched networks. Openflow [2] provides a key SDN building-block by defining a standardized protocol for a controller to interact with switches: installing *rules* for forwarding and modifying packets, querying traffic counters, and directing packets to a *controller* for further processing. Today's controller applications manage the network at the level of individual switches. Yet, a truly powerful SDN system should enable reasoning about the rules installed across *multiple* switches, and their cumulative effect on packets traversing the network.

The goal of our work is to address this challenge by building machinery for *global policy transformation*. Global policy transformation operates by moving, merging, and/or splitting rules across multiple switches, while preserving the overall forwarding behavior of a network. Using this machinery, policies written for a single switch can be distributed across a set of switches, as shown in Figure 1, and policies already distributed over multiple switches can be consolidated. This is important for several reasons:

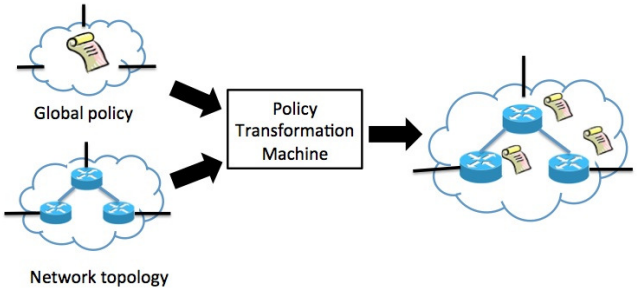

Figure 1: Policy transformation.

- Resource usage: Policies too large to fit on any one switch can be spread over multiple switches.
- Ease of programming: Policies for a single switch are easier for programmers to write.
- Portability: Policies written for one network topology can be transformed to run on another.
- Analysis: A policy spread over multiple switches can be transformed into a single list of rules, to more easily check that the policy satisfies invariants.

Our goal is to enable SDN controllers to transform policies automatically, shielding the programmer from the details.

For many years, researchers have studied firewalls represented as a prioritized list of access-control rules that permit or deny traffic, with the goal of checking whether two policies are equivalent, or to minimize the number of rules. For instance, Appelgate *et al.* [1] show that minimizing a single access control list is NP hard, and present an optimal algorithm for one particular case of access-control lists, while Yuan *et al.* [4] analyze the correctness of distributed firewalls. Other related work [3] develop general algorithms to optimize a single list. Our work is the first to examine the general problem of network-wide *policy transformation*.

We identify two critical challenges with respect to *policy transformation*. The first is *correctness*: each packet `p` must be treated under a transformed policy `A'`, exactly as it would under the original policy `A`. The second is that of *efficiency*: transformed policies should minimize the total number of rules, minimize the maximum number of rules on any switch, or, in the case of a network of switches with different capacities, find a way to pack rules in to differ-

SIGCOMM'12, August 13–17, 2012, Helsinki, Finland.
ACM 978-1-4503-1419-0/12/08.

ent switches so that no switch overflows its capacity. We plan to address these problems of correctness and efficiency by developing a general-purpose policy rewriting framework that characterizes all possible, sound policy transformations. This framework will provide a formal foundation that can be used to analyze the correctness of specific policy rewriting algorithms.

2. POLICY TRANSFORMATION

We begin our exploration of global network policy optimization by analyzing two categories of network policy: the single switch policy and the chain-of-switches policy. Other topologies are certainly interesting, but as the topology grows in complexity, so too grows the complexity of the transformation system. We continue by discussing the rewriting axioms that are necessary and sufficient to convert from one policy to another (in the same category). We conclude with a brief mention of our ongoing work to use these rewriting systems to develop efficient and correct deterministic optimization algorithms.

2.1 Policy Categories

A policy is a collection of packet-forwarding rules where each rule r is a $(pattern, action)$ pair. Typical patterns include exact-match patterns, wildcard patterns and prefix-match patterns. Since any pattern p will match some set of packets (perhaps a singleton set), we write $p \subseteq p'$ when p matches a smaller set of packets than p'. Typical actions include forward, drop, and modify actions.

A *single-switch policy* is a list $l = r_1, r_2, \ldots, r_n$ of rules. The rules appear in the list with decreasing priority. Therefore, to compute the action for a packet h in a rule list l, one can linearly scan l from left to right until hitting a rule (p, a) where the packet header h matches p. In such a case, the action a is applied to the packet.

A *chain-of-switches policy* c is organized as a list of single-switch policies (*i.e.*, $c = l_1; l_2; \ldots; l_m$). Switch chains can be used to model both cascading flowtables within a switch as well as a set of switches placed in series.

In general, when describing a generic policy we use an upper-case letter P. The semantics of a policy P relative to a packet h, written $P[h]$, is the set of observable actions applied by the policy to that packet as it traverses the network.

2.2 Policy Rewriting

Policy rewriting is specified via a collection of simple, local rewriting axioms. These simple local axioms may be composed with one another to produce arbitrary semantics-preserving policy transformations. As an example, consider the Shadow axiom below.

$$\frac{p_2 \subseteq p_1}{(p_1, a_1), (p_2, a_2) \leftrightarrow (p_1, a_1)} \quad (Shadow)$$

One should read the statements above the horizontal line as assumptions and statements below the line as conclusions. Hence, the axiom states that if pattern p_2 is a subset of pattern p_1 then the pair of adjacent rules $(p_1, a_1), (p_2, a_2)$ can be rewritten to the single rule (p_1, a_1), and vice-versa.[1] Such local rewrites may be applied correctly in a large context using the Congruence axiom as follows.

$$\frac{l_2 \leftrightarrow l_2'}{l_1, l_2, l_3 \leftrightarrow l_1, l_2', l_3} \quad (Congruence)$$

The Congruence axiom states that if a local rewrite applies then that local rewrite may be used within the context of a longer list of rules. In addition to Shadow and Congruence, there are a few other axioms for single switch policies including Reordering and Join.

Rewriting rules for switch chains relies on the rewriting axioms for the single-switch case to optimize individual switches in the chain. In addition, there are further axioms for shuffling functionality back and forth along the chain.

2.3 Soundness and Completeness Properties

Any good rewriting system should satisfy two key properties: (1) A rewriting system is *Sound* if any combination of rewriting axioms is guaranteed to produce a semantically equivalent list of rules, and (2) a rewriting system is *complete* if any semantically equivalent policy can be produced by applying the axioms. More precisely:

Definition 1. Soundness: If $P_1 \leftrightarrow P_2$ then for all packet headers h, $P_1[h] = P_2[h]$.

Definition 2. Completeness: If for all packet header h, $P_1[h] = P_2[h]$, then $P_1 \leftrightarrow P_2$.

To date, we have proven our rewrite system for single-switch policies is sound and complete and our rewrite system for switch-chains is sound. In the future, we hope to develop a completeness argument for switch-chain rewriting. We are also looking at richer topologies including trees and DAGs.

2.4 Ongoing Work: Optimization Algorithms

Our rewrite system does not itself constitute a deterministic algorithm for generating optimized policies. Rather, it describes the search space any such algorithm must navigate, and provides a framework within which one may analyze the correctness of such algorithms.

The next phase of the research involves developing multi-phase algorithms for generating optimized, global network policies. Our current plan involves three phases: (1) a heuristic algorithm to distribute global policy to particular switches, (2) balancing across neighbors via switch-to-switch rule movement and (3) optimization on individual switches.

[1]Note that sometimes an optimization is enabled by first expanding a rule in to multiple rules that are easier to manage or move around.

3. REFERENCES

[1] D. L. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang. Compressing rectilinear pictures and minimizing access control lists. In *ACM-SIAM SODA*, pages 1066–1075, 2007.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Computer Communications Review*, 38(2):69–74, Mar. 2008.

[3] C. R. Meiners, A. X. Liu, and E. Torng. TCAM Razor: A systematic approach towards minimizing packet classifiers in TCAMs. *IEEE/ACM Trans. Netw.*, 18(2):490–500, Apr. 2010.

[4] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra. FIREMAN: A toolkit for firewall modeling and analysis. In *IEEE Symposium on Security and Privacy*, pages 199–213, 2006.