Propane: Programming Distributed Control Planes

Extended Abstract (demo)

Ryan Beckett
Princeton University
rbeckett@cs.princeton.edu

Ratul Mahajan
Microsoft Research
ratul@microsoft.com

Todd Millstein
University of California, Los Angeles
todd@cs.ucla.edu

Jitendra Padhye
Microsoft Research
padhye@microsoft.com

David Walker
Princeton University
dpw@princeton.edu

ABSTRACT

We describe Propane, a new system with a language for specifying the end-to-end routing behavior of the network and a compiler for implementing the policies using the a collection of device configurations for the BGP routing protocol that run on unmodified vendor hardware. Propane allows operators to describe their policy through high-level constraints on both the shape and relative preferences of paths for different types of traffic. Constraints can describe paths both through the user's network, as well as through other networks that are not directly under the user's control. When Propane compiles a policy, the resulting BGP configurations are guaranteed to correctly implement the centralized policy in a distributed fashion – without any centralized coordination and regardless of any number of network failures. We will demo Propane by showing how to write and compile a simple data center routing configuration.

1 INTRODUCTION

Operators of traditional networks have a difficult job: They are charged with achieving a wide variety of network-wide objectives, but they must do so by managing and configuring many individual, separate devices, each of which runs complex distributed control plane protocols. However, existing languages for network configuration provided by vendors are too low level, requiring operators to work with assembly-language-like configuration primitives such as route-filters, community tags, and local-preference among routes. Further, the distributed nature of the network makes implementing network-wide policy more difficult. Many policies involve network-wide properties, for example to prefer a certain neighbor, keep certain traffic in a geographic region, or use a particular path only as a backup – yet configurations describe the behavior of individual devices.

It is up to the operator then to perform the intellectually challenging task of decomposing a network-wide policy into a collection of distributed device policies such that their distributed interactions emulate the original policy.

To make matters worse, the possibility for network failures is an ever-present concern that significantly complicates the operator's task. Reasoning about all possible combinations of failures and their potential impact on a distributed system is simply too hard for most humans. It is not surprising then that configurations that work correctly in fault-free environments, have been found to result in incorrect behavior when network elements go down [4]. As a consequence of the difficulty of achieving network-wide objectives through device-level configuration, in practice, configuration errors are responsible for a large fraction of highly disruptive network outages [1, 3–7, 9].

In recent years, Software-defined networking (SDN) has emerged as an alternative approach to traditional network configuration. Rather than organizing routing policy around a collection of configurations that compute routes via well-known distributed protocols, users can directly program the forwarding tables of SDN-enabled switches via a centralized controller. It is the job of the controller then, which has a centralized view of the network to make sure that the routing policy is implemented correctly.

However, the SDN approach comes with its own set of challenges. For one, centralized SDN systems introduce a potential single point of failure and thus must be carefully designed with robustness in mind. The controller(s) must always be able to communicate with every switch even when failures occur, or if the network is large and geographically-distributed. Furthermore, such systems must also be carefully engineered for low-latency and scalability since the centralized controller can quickly become a potential performance bottleneck. For many of these challenges, researchers have looked at systems with multiple, interacting controllers, thus bringing back some aspects of distributed control planes [2, 8]. Second, current SDN systems focus only on configuration of intra-domain routing policy, forcing users to fall back to traditional methods for configuring inter-domain routing policy. More pragmatically, many networks will continue to use a distributed control plane for the foreseeable future, due to the difficulty of migrating to SDN or the inherent scalability and failure-robustness of distributed control.

Language Overview. We will demo our recent efforts to design and implement a new language called Propane, which greatly simplifies the task of network management by allowing operators to specify their objectives using high-level, *end-to-end* constraints on forwarding paths rather than writing explicit device-by-device configurations.

The Propane compiler rather than a human is responsible for bridging the gap between high-level goals and the low-level mechanisms. More specifically, Propane allows operators to describe the kinds of paths traffic may or may not follow, as well as the relative preferences of such paths. Paths can reference both devices in the network as well as other networks that are not under the control of the operator.

Taken together, these constraints allow users to write rich routing policies for the network. For example, then can require that two nodes be reachable, that traffic never leaves a particular part of the network, or that traffic is waypointed through middleboxes. Preferences allow operators to write policies that describe desired behavior after network failures occur, such as preferring to use routes through one neighboring network over another, or using particular paths only as backups for other paths.

The Propane language also supports network operators by defining a set of common abbreviations for entering, leaving, and traversing the user network and provides abstractions for managing common features of control plane algorithms such as route aggregation. Operators use familiar logical operations to build routes in a modular and compositional manner from these primitives.

Compiler Overview. Propane analyzes these user specifications and then compiles them to a collection of BGP configurations, so they may exploit existing distributed policy and fault tolerance mechanisms and execute on existing commodity hardware. During the compilation process, the compiler automatically synthesizes low-level configuration primitives such as per-device import and export filters as well as BGP local preferences, MED attributes, and community tags. The compiler accomplishes this by first transforming the policy into a graph-based intermediate representation that is more amenable to compilation and analysis.

An important feature of the language is that if possible, compiled configurations are guaranteed to implement the correct routing policy, regardless of any number of failures that might occur in the network. If correct compilation is impossible, the compiler notifies the user at compile time rather than waiting until the system is deployed in operation and failures have occurred in the network.

This guarantee does not mean that the implementation is always able to send traffic to its ultimate destination (e.g., in the case of a network partition), but rather that it always respects the centralized policy, which may include dropping traffic when there is no route.

Overall, we believe the approach Propane combines easy programmability of centralized control planes with the failure robustness and scalability of distributed control planes.

Demo Overview. The Propane demo will give a brief overview of the language and compiler by demonstrating how to write and compile a simple Propane policy for a data center network. We will then demonstrate how the generated configurations can be emulated using Quagga router software with the open-source CORE network emulator. An overview of what the demo can look like can be found here:

https://drive.google.com/open?id=0B-rjACAtTwE_c0JGczdNRFBYcEE

Acknowledgments. We thank R. Aditya, George Chen, and Lihua Yuan for feedback on the work and the SIGCOMM reviewers for comments on the paper. This work is supported in part by the National Science Foundation awards CNS-1161595 and CNS-1111520 as well as a gift from Cisco.

REFERENCES

[1] M. Anderson. Time warner cable says outages largely resolved. http://www.seattletimes.com/business/time-warner-cable-says-outages-largely-resolved, August 2014.

[2] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar. ONOS: Towards an open, distributed SDN OS. In *HotSDN*, August 2014.

[3] N. Feamster and H. Balakrishnan. Detecting BGP configuration faults with static analysis. In *NSDI*, May 2005.

[4] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein. A general approach to network configuration analysis. In *NSDI*, March 2015.

[5] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: Measurement, analysis, and implications. In *SIGCOMM*, August 2011.

[6] Z. Kerravala. What is behind network downtime? proactive steps to reduce human error and improve availability of networks. https://www.cs.princeton.edu/courses/archive/fall10/cos561/papers/Yankee04.pdf, January 2004.

[7] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *SIGCOMM*, August 2002.

[8] J. McCauley, A. Panda, M. Casado, T. Koponen, and S. Shenker. Extending SDN to large-scale networks. In *Open Networking Summit*, April 2013.

[9] J. Networks. As the value of enterprise networks escalates, so does the need for configuration management. https://www-935.ibm.com/services/au/gts/pdf/200249.pdf, May 2008.