Abstract— We develop Propane, a language and compiler to help network operators with a challenging, error-prone task—bridging the gap between network-wide routing objectives and low-level configurations of devices that run complex, distributed protocols. The language allows operators to specify their objectives naturally, using high-level constraints on both the shape and relative preference of traffic paths. The compiler automatically translates these specifications to router-level BGP configurations, using an effective intermediate representation that compactly encodes the flow of routing information along policy-compliant paths. It guarantees that the compiled configurations correctly implement the specified policy under all possible combinations of failures. We show that Propane can effectively express the policies of datacenter and backbone networks of a large cloud provider; and despite its strong guarantees, our compiler scales to networks with hundreds or thousands of routers.

1 INTRODUCTION

It is well known that configuring networks is error prone and that such errors can lead to disruptive downtimes [6, 8, 10, 13]. For instance, a recent misconfiguration led to an hour-long, nation-wide outage for Time Warner's backbone network [2]; and a major BGP-related incident makes international news every few months [4].

A fundamental reason for the prevalence of misconfigurations is the semantic mismatch between the intended high-level policies and the low-level configurations. Many policies involve network-wide properties—prefer a certain neighbor, never announce a particular destination externally, use a particular path only if another fails—but configurations describe the behavior of individual devices. Operators must manually decompose network-wide policy into device behaviors, such that policy-compliant behavior results from the distributed interactions of these devices. Policy-compliance must be ensured not only under normal circumstances but also during failures. The need to reason about all possible failures exacerbates the challenge for network operators. As a result, configurations that work correctly in failure-free environments have nonetheless been found to violate key network-wide properties when failures occur [8].

To reduce configuration errors, operators are increasingly adopting an approach in which common tasks are captured as parameterized templates [12, 17]. While templates help ensure certain kinds of consistency across devices, they do not provide fundamentally different abstractions from existing configuration languages or bridge the semantic divide between network-wide policies and device-level configuration. Thus, they still require operators to manually decompose policies into device behaviors.

As a complementary approach, configuration analysis tools can help reduce misconfigurations by checking if low-level configurations match high-level policy [6, 8]. However, such tools cannot help operators with the challenging task of generating configurations in the first place.

Software-defined networking (SDN) and its abstractions are, in part, the research community's response to the difficulty of maintaining policy compliance through distributed device interactions [5]. Instead of organizing networks around a distributed collection of devices that compute forwarding tables through

mutual interactions, the devices are told how to forward packets by a centralized controller. The controller is responsible for ensuring that the paths taken are compliant with operator specifications.

The centralized control planes of SDN, however, are not a panacea. First, while many SDN programming systems [9] provide effective *intra*-domain routing abstractions, letting users specify paths within their network, they fail to provide a coherent means to specify *inter*-domain routes. Second, centralized control planes require careful design and engineering to be robust to failures—one must ensure that all devices can communicate with the controller at all times, even under arbitrary failure combinations. Even ignoring failures, it is necessary for the control system to scale to meet the demands of large or geographically-distributed networks, and to react quickly to environmental changes. For this challenge, researchers are exploring multi-controller systems with interacting controllers, thus bringing back distributed control planes [3, 14] and their current programming difficulties.

Hence, in this paper, we have two central goals:

(1) Design a new, high-level language with natural abstractions for expressing intra-domain routing, inter-domain routing and routing alternatives in case of failures.
(2) Define algorithms for compiling these specifications into configurations for devices running standard distributed control plane algorithms, while ensuring correct behavior independent of the number of faults.

To achieve the first goal, we borrow the idea of using regular expressions to specify network paths from recent high-level SDN languages such as FatTire [15], Merlin [16], and NetKAT [1]. However, our design also contains several key departures from existing languages. The most important one is semantic: the paths specified can extend from outside the operator's network to inside the network, across several devices internally, and then out again. This design choice allows users to specify preferences about both external and internal routes in the exact same way. In addition, we augment the algebra of regular expressions to support a notion of *preferences* and provide a semantics in terms of sets of ranked paths. The preferences indicate fail-over behaviors: among all specified paths that are still available, the system guarantees that the distributed implementation will always use the highest-ranked ones. Although we target a distributed implementation, the language is more general and could potentially be used in an SDN context.

To achieve the second goal, we develop program analysis and compilation algorithms that translate the regular policies to a graph-based intermediate representation and from there to per-device BGP configurations, which include various filters and preferences that govern BGP behavior. We target BGP for pragmatic reasons: it is a highly flexible routing protocol, it is an industry standard, and many networks use it internally as well as externally. Despite the advent of SDN, many networks will continue to use BGP for the foreseeable future due to existing infrastructure investments, the difficulty of transitioning to SDN, and the scalability and fault-tolerance advantages of a distributed control plane.

The BGP configurations produced by our compiler are guaranteed to be policy-compliant in the face of *arbitrary* failures.[1] 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. In this way, we provide network operators with a strong guarantee that is otherwise impossible to achieve today. However, some policies simply cannot be implemented correctly in BGP in the presence of arbitrary failures. We develop new algorithms to detect such policies and report our findings to the operators, so they may fix the policy specification at compile time rather than experience undesirable behavior after the configurations are deployed.

We have implemented our language and compiler in a system called Propane. To evaluate it, we use it to specify real policies for datacenter and backbone networks. We find that our language expresses such policies easily, and that the compiler scales to topologies with hundreds or thousands of routers, compiling in under 9 minutes in all cases.

2 CONCLUSIONS

We introduced Propane, a language and compiler for implementing network-wide policies using a distributed set of devices running BGP. 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. When Propane compiles a policy, the resulting BGP configurations are guaranteed to implement the centralized policy in a distributed fashion, regardless of any number of network failures. Applying Propane to real-world networks showed that its language is expressive and its compiler is scalable.

REFERENCES

[1] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker. NetKAT: Semantic foundations for networks. In *POPL*, January 2014.

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

[3] 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.

[4] News and press | BGPMon. http://www.bgpmon.net/news-and-events/.

[5] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. In *SIGCOMM*, August 2007.

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

[7] N. Feamster, J. Winick, and J. Rexford. A model of BGP routing for network engineering. In *SIGMETRICS*, June 2004.

[8] 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.

[9] N. Foster, M. J. Freedman, A. Guha, R. Harrison, N. P. Katta, C. Monsanto, J. Reich, M. Reitblatt, J. Rexford, C. Schlesinger, A. Story, and D. Walker. Languages for software-defined networks. *IEEE Communications Magazine*, 51(2):128–134, February 2013.

[1] We assume that BGP is the only routing protocol running in the network or the other protocols are correctly configured and do not have adverse interactions with BGP [7, 11].

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

[11] T. G. Griffin and G. Wilfong. On the correctness of IBGP configuration. In *SIGCOMM*, August 2002.

[12] Hatch – create and share configurations. http://www.hatchconfigs.com/.

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

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

[15] M. Reitblatt, M. Canini, N. Foster, and A. Guha. FatTire: Declarative fault tolerance for software defined networks. In *HotSDN*, August 2013.

[16] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster. Merlin: A language for provisioning network resources. In *CoNEXT*, December 2014.

[17] configuration templates | thwack. https://thwack.solarwinds.com/search.jspa?q=configuration+templates.