NV: An intermediate language for network verification

Introduction. Network devices often rely on distributed protocols, such as BGP, to make routing decisions. Network operators can enforce routing policies (that may express security, economic or other concerns) by configuring what routing protocols each device executes, how it uses the routing information received by neighbors and what routing information it shares. These configurations are expressed in low-level, vendor specific languages. Combined with the distributed nature of routing protocols, reasoning about the correctness of the configurations – and in extension, of the network – is a daunting task for operators. Network verification [1, 4] and simulation tools [3] have been proposed to aid operators. Additionally, as those techniques often face scaling problems, researchers have suggested network transformations [2, 5] in order to simplify the complexity of networks.

Modelling network configurations. Irregardless of the transformation or reasoning principles used, one needs to parse the original network configurations as provided by operators. Batfish [3] was the first to tackle this problem, by devising a vendor-agnostic representation of common routing protocols and providing a translation from each vendor's language to the Batfish representation, a concept akin to intermediate languages, such as LLVM, in compilers. Subsequent analysis such as compression [2], simulation [3] or verification [1, 4] can be then performed on top of this representation.

Batfish has been an indispensable tool for network researchers thanks to its ability to parse a wide range of configurations from different vendors. Unfortunately, its intermediate language (IR) falls short of many language design goals. First, at 105 different expressions and 23 statements Batfish's IR is *massive*. This is a symptom of other problems in the design of the IR. In particular, the expressions and structures used are highly *specialized* to common routing protocols (e.g. expression to set the local preference of a BGP attribute). As such, they cannot be *composed* to build other operations. Besides the explosion in the size of the IR, this poses another issue: desirable transformations often cannot be expressed within Batfish's IR. For instance, replacing the `AS Path` attribute of BGP with its length, can often improve simulation performance without loss of precision. Yet, this simple transformation cannot be expressed within Batfish's current AST. Moreover, the semantics of the language requires a deep understanding of routing protocols and the intricacies of vendor specific implementations to understand. As a consequence, because of the size of the language and the complexity of its semantics, writing new analyses over configurations is often a tedious task.

Finally, Batfish suffers from the fact that some effects of executing a protocol are not expressed in the configurations, but are left *implicit* and it's up to the backend (simulator, SMT verifier, etc.) to correctly capture them. This makes it notoriously difficult to analyze configurations, one has to look at the backend to understand the semantics of configurations. It also makes it harder to keep the various backends in sync.

NV: A flexible IR for control plane configurations. To overcome these limitations, we propose an intermediate language, called NV, that is based on the ML programming language [6]. As the basis of many modern programming languages, ML has been widely studied and has a *rigorous semantics*. As a result, one does not need to be an expert in routing protocols to understand an NV program. Importantly, we demonstrate that with just *a few* standard constructs (options, pairs, booleans/integers and their operations, functions, dictionaries, graphs, match statements) we can clearly express the configurations of common routing protocols such as BGP and OSPF, by translating the Batfish IR to NV.

Implementing and reasoning about the correctness of transformations of NV code is facilitated by the small size of the language and the simplicity of its semantics. For instance, it is straightforward to implement common compiler transformations such as constant folding and inlining, to improve the performance of simulation/verification. Or a transformation where the path component of BGP is replaced by an integer. Moreover, because the semantics of the program remain explicit, those transformations do not require any change in subsequent tools in the toolchain (such as a Batfish style simulator, or an SMT verifier).

Challenges. When defining a new IR there is a tension between expressiveness of the language, i.e. the ability to accurately describe common routing protocols and potentially new ones, and the efficiency of simulation/verification techniques. For instance, total maps is an essential data structure in the context of networks, but it does not admit an efficient or complete SMT encoding. To achieve the desired efficiency and completeness goal of our SMT encoding of the IR, we had to impose additional restrictions on the operations of the maps. In a general purpose language these restrictions

may not make sense, but in the context of routing protocols they do not hinder expressiveness.

On a similar note, Batfish's specialized expressions provide room for specialized (in terms of efficiency) interpretations. For example, when simulating a network that runs iBGP over some IGP, Batfish will not simulate the iBGP part until the IGP has converged. This and other similar optimizations help speed up the simulation. In an IR where iBGP and IGP are not directly exposed, but rather are build from smaller blocks, it is not straightforward to implement such optimizations. Exploiting the high-level structure of the configurations to – among other things – improve the performance of the simulation is an important future direction.

* Provenance? Pinpointing to the fault may not be as easy..

REFERENCES

[1] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. A general approach to network configuration verification. In *SIGCOMM*, August 2017.

[2] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. Control plane compression. SIGCOMM '18, pages 476–489, 2018.

[3] 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*, 2015.

[4] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan. Fast control plane analysis using an abstract representation. In *SIGCOMM*, 2016.

[5] N. Giannarakis, R. Beckett, R. Mahajan, and D. Walker. Efficient verification of network fault tolerance via counterexample-guided refinement, 2019.

[6] R. Milner. A theory of type polymorphism in programming. *Journal of computer and system sciences*, 17(3):348–375, 1978.