Demo: Structural Network Minimization: A Case of Reflective Networking

Mubashir Anwar[†], Anduo Wang[*], Fangping Lan[*], Matthew Caesar[†]
[†]University of Illinois at Urbana-Champaign, [*]Temple University

ABSTRACT

Traditional network state management focuses on packets that exercise network structures (configurations, procedures) and testify semantics (intentions), but provides little insights into how the structure actually "causes" the semantics. In response to this missed opportunity, we propose *reflective networking*, which features *a network structure capable of altering itself with a causal connection to its semantics*. Specifically, we investigate the network datalog structure and the chase, a process that transforms datalog programs by "executing" intents (semantic constraints) that are themselves expressed in datalog. To illustrate the usefulness of reflective networking, this demonstration presents a first use case: we developed an intuitive specification of routing in datalog, and employed the chase to summarize a network's routing behavior by minimizing (repeatedly transforming) the corresponding datalog program.

CCS CONCEPTS

• **Networks** → *Programming interfaces*; *Network manageability*; • **Computing methodologies** → *Reasoning about belief and knowledge*.

KEYWORDS

Network management, reflective programming, datalog minimization, the chase

1 INTRODUCTION

Managing network state — the network structure (configurations and procedures) and the semantics (intentions) — has been challenging [**? ? ?**] since the early days of the Internet, when computer networks were still a bag of distributed protocols. Management in general was a black art: The admin often made small changes to the network and used primitive tools (ping, traceroute) to check the resulting outcome on some test (witnessing) packet. Such trial-and-error methods hardly generalize. For example, after verifying a specific intention or fixing a particular anomaly by exercising relevant packets (drawn from past experience, perhaps), very little insights are generated into the network state in general: Which (set of distributed) devices or parameters are responsible for causing what property? What modification (adding which link, altering which command) corrects which anomaly? Can we transfer the logic handling interdependencies on one network into another even when the difference is tiny?

As networks become more programmable (software-defined), we have SDN controllers that eliminate distributed state management, domain-specific languages that raise the level of abstraction, and formal verification and synthesis tools that are far more powerful than primitive tools like ping. Combined together, do they entail a better management of the network state? The SDN software, though centralized and higher-level, tend to be very expressive and the problem of managing the intention, as buried in a software, is notoriously hard. The formal reasoning tools, despite involving sophisticated techniques (symbolic execution, model abstraction, search heuristics etc), still follow the same old *behavioral approach* in which a network state is examined by exhaustively trying out all possible inputs/outputs packets, revealing very little about how and why an intention is collectively implemented by the network structure as a soup of software expressions. In fact, as shown by the test coverage metric (proposed in [**? ?**]) that measures the fragment of network rules being exercised, even our best assessment of the network state is primitive.

In light of these, we take a departure from the traditional approach and propose a *reflective* [**?**] one in which network state is managed by *a network structure capable of examining and altering itself with a causal connection to the behavior (semantics).* Reflective networking shifts the focus from input/output packets analysis to the network structure itself, to its ability to *reflect* on questions including: Can we identify the part of a network structure (a data, an operation, a line of code) relevant to an intention, and alter it to match a change? Can we test containment of network behaviors by syntactic (structural) comparison, not a verbatim comparison for specific scenarios, but generally? Can we have a true notion of network optimality, some "smallest" structure for the same behavior, and have a procedure to actually find it?

As a first step towards this vision, we make a case of reflective networking with the datalog structure. Datalog-like languages (referred to as network datalog), have been gaining increasing popularity in SDN platforms [**? ? ?**] and verification tools [**? ? ?**], but we choose datalog because it is by nature, though less well exploited, reflective. Datalog is closely related to a database query language called *tableau* [**?**] (plural: tableaux), a visual data query — similar to Microsoft Access and Query-By-Example (QBE [**?**]) — that specifies a query by giving a representative example — a "template" table with both constant and variable symbols — that cover all possible answers. A tableau can thus be viewed as both a program (query) and a data instance (template table). This *program-data duality* carries over to datalog too. We only need to view the body of a datalog program (for datalog rule $h :- b_1, \cdots, b_n$, its body is the set of predicates $b_1, \cdots, b_n$ with constants and variables) as a symbolic database — the variables in this database now describe all

ACM SIGCOMM '23, September 10, 2023, New York, NY, USA

ACM ISBN 979-8-4007-0236-5/23/09.
https://doi.org/10.1145/3603269.3610847

databases that can derive an answer (the head h). Leveraging this program-data duality, as formalized by the elegant chase process [**? ? ?**], we can use the familiar (e.g., SQL-like) data operations to examine and modify datalog programs, including testing if one datalog program's behavior contains another, computing how a semantic constraint (so called data dependencies) affects and alters a program, and optimizing a datalog program — by repeatedly testing if eliminating any part of the program retains its original semantics[1] — until a minimal equivalent is reached. The chase thus posses the essential ingredients for all the example reflections we envision for reflective networking.

But the classic chase, even in its most recent development [**?**], is only effective for a limited class of data dependencies that do not fully capture the complex semantics of networking. So our first contribution is to extend the classic chase theory to networking. We analyzed and identified the root causes to classic chase's limitation: the chase uses the standard query evaluation on a datalog program's instantiated database. The instantiated database, however, is a symbolic one, or technically, an incomplete database that requires more advanced evaluation. Based on this insight, our poster will illustrate a new chase algorithm that supports the incomplete database structure and incorporates the *fauré-log* evaluation engine, a network datalog extension for incomplete information. In addition, our demonstration illustrates the feasibility of reflective networking by presenting an empirical study of network minimization with the classic chase. In particular, we designed a stress test — summarizing arbitrary synthetic routing services on Rocketfuel [**?**] topologies with Route Views [**?**] prefixes by minimizing the corresponding network datalog model, and developed a prototype of the classic chase. Our experience is promising: the chase enables general network minimization (asymmetric topology, extensible to firewall and middleboxes) with competitive performance (< 1 minute even for minimizing large services with our most computationally expensive model), uncovers interesting findings into ISP redundancy, making it a valuable platform to rigorously and quantitatively manage network services.

2 DEMONSTRATION PLAN

In our demonstration, we will use a concrete example to introduce reflective networking as an attractive alternative to popular state-of-the-art tools [**? ? ?**], and to illustrate how reflective networking naturally generalizes previous heuristics [**? ?**].

An example: summarizing services

Restricted forms of structural manipulation of networks do exist in earlier work. One notable example is network transformation, a technique proposed to scale up existing packet-centric verifiers [**? ?**]. Note that network transformation is a specific verification task that establishes network equivalence (e.g., decide if a larger network is equivalent to a smaller one while preserving path properties); making transformation, by definition, is as (computationally) expensive as verification. How can we scale up the later by the former then? The answer by existing work is to only look for those "efficient" transformations by exploiting *structural properties* already presented in the network. One such structural property is topology symmetry in data center networks [**?**], as depicted in Figure 1 (a). Moving beyond such structural properties, however, existing methods quickly degenerate into simple (if not trivial) comparison, for example, the Campion debugger [**?**] considers two OSPF configuration equivalent if the OSPF costs are identical.

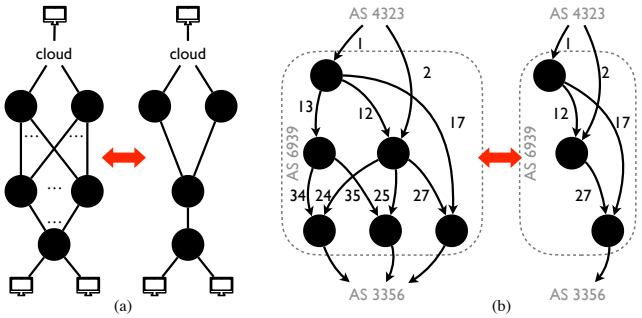

Figure 1: (a) Effective behavioral network transformation relies on topology symmetry (a structural property); (b) Can we generalize to arbitrary topology with structural transformation.

By contrast, reflective networking seeks general, non-trivial network transformation. To see a potential use, consider the (actual) Rocketfuel topology for `AS6939` [**?**] shown in Figure 1 (b): on the left, the 6 nodes belonging to `AS6939` provide various paths for `AS4323` to reach `AS3356`. When considering only reachability and path properties, it is commonly believed that the reachability services offered in the global internet, as demonstrated in this example, have redundancies. In fact, with some manual examination, it is not hard to see that the smaller network on the right gives an optimal summary in the sense that no more link can be removed to retain the same diversity (reachability, path length). To achieve this, one challenge is to properly encode the "redundancy", requiring an understanding of both how structural minimization operates and what semantics we want to retain. Besides, minimization only gives the end product — a minimized program that models the compressed network, it does not report the source of redundancy. To this end, we also developed a novel companion process (augmenting our model) that pin-points the redundancy.

A solution: structural network minimization We developed a datalog engine using Python and integrated it with *fauré* [**?**] evaluation, utilizing PostgreSQL [**?**] as the underlying query evaluation engine. By leveraging *fauré-log*, we are able to effectively represent incomplete information, which enhances the expressiveness of our network models. Furthermore, we optimized and implemented program minimization for *fauré-log*. The demonstration will showcase (1) the application of datalog encoding to the reachability services offered by ASes, similar to the example in Figure 1, and (2) various network encodings in datalog that preserve different network properties (e.g. middle boxes and firewalls) during program minimization.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation Award CNS-1909450, CNS-2145242.

[1] Static query optimization was actually the early killer app that drove the development of the chase [**?**].