Sarasate: A Strong Representation System for Networking Policies

Bin Gui, Fangping Lan, Anduo Wang
Temple University

ABSTRACT
Policy information in computer networking today are hard to manage. This is in sharp contrast to relational data structured in a database that allows easy access. In this demonstration, we ask why cannot (or how can) we turn network policies into relational data. Our key observation is that oftentimes a policy does not prescribe a single “definite” network state, but rather is an “incomplete” description of all the legitimate network states. Based on this idea, we adopt conditional tables and the usual SQL interface (a relational structured developed for incomplete database) as a means to represent and query sets of network states in exactly the same way as a single definite network snapshot. More importantly, like relational tables that improve data productivity and innovation, relational policies allow us to extend a rich set of data mediating methods to address the networking problem of coordinating policies in a distributed environment.

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

KEYWORDS
Network policies, c-tables, knowledge representation

1 INTRODUCTION
Policies are a fundamental part of computer networking, and many tools have been developed to exploit policies and/or manage them throughout a network’s entire life cycle. Notably, higher-level programming abstractions help the operators to realize reachability objectives in SDNs [10, 15, 16, 19, 20, 22, 23]; intention-aware monitoring systems leverage network-wide queries to improve monitoring in programmable hardware [7, 28]; BGP configurations [4, 9, 13, 24, 26] are still the main vehicle for affecting routing — whether in the global Internet or datacenters — whenever rich semantics is involved; verification tools [3, 29] with varying capabilities — expressiveness, scalability, speed — check whether the network configurations, SDN programs, etc. actually obey the properties in the formal specification; and synthesizers [4, 5] attempt to convert network specification of varying forms — e.g., logical assertions, templates — directly into a concrete implementation.

Yet using and managing networking policies remain hard: One has to fully understand the protocol mechanisms or its operating environment to properly set policy attributes in that protocol; In SDNs, while the goal is to simplify management, the lack of prefixed mental model makes it more difficult for anyone who is not involved in writing the controller program in the first place to make sense of or debug a policy: Besides the tight coupling with network systems, in more automated tasks like formal verification or synthesis, policies also take on the characteristics — expression in a specialized logic or format — of that external synthesizer or verifier that require significant expertise. And few in the networking community question the need for increasingly more complex and disparate structures of policies, or the deeper integration of policies with the rest of the system(s).

This is in contrast to data management in relational database [1] which is notably easier: The relational data model has replaced many non-relational application-specific data structures, it is self-explaining and gives a common understanding of the data, thus allowing for communication across tasks and between users; The relational database system, in addition, while striking a promising trade-off point between specification complexity and performance, does not intend to be a total solution, instead, it draws a clean boundary between a shared data component and the external applications, exposes to applications an intuitive yet rigorous (SQL) interface, thus improving productivity of application programmers, enabling independent evolutions, and accelerating innovations. Where networking today requires people to master more computer science skills, database has already been successfully running for non-programmers with little expertise.

So why cannot (how can) we turn network policies into data structured in a database? One difficulty with a genuine data approach is that a policy is rarely captured in a definite network snapshot. While existing networking approaches address this by including the broader contexts, e.g., mechanisms, dynamics, additional models, etc., to fully express a policy, we argue that it is probably adequate to represent a policy by all the possible network states it admits, just like a predicate in set theory is accurately described by the set it admits, just like a predicate in set theory is accurately described by the set it corresponds to. Based on this idea, we propose relational policies, capable of representing and processing sets of possible network states in exactly the same way as that of a network snapshot. Central to relational policies is a relational structure called conditional tables [1, 2, 14] which extend regular tables with variables and constraints over those variables, which are operated via an interface that is both intuitive — the familiar SQL operations, and are rigorous — SQL operations on conditional tables are safe (deriving some information only if it is really in some legitimate network state) and complete (capture all the legitimate states).

Like relational databases, relational policies seek to provide a versatile shared policy component capable of rapid innovations, but with the added benefit of exploiting a wealth of solutions already
developed in relational databases. Specifically, we use relational policies to study two concrete problems: (1) local policy making can approximate global optimal by participating in some form of information exchange [8, 11, 12, 17, 18, 27] — but the exchange is often low-level and ad hoc, and (2) most policy-rich tools enforce, check, synthesize a set of coherent policies [3, 29] — but says little about how to obtain a consistent policy from disparate sources (e.g., teams overseeing overlapping aspects or interacting parts of the network) in the first place.

This demo presents sarasate, a prototype realization of our vision atop PostgreSQL (the world’s most advanced open source relational database) [21] and Z3 [6] (a state-of-the-art constraint solver popular in networking). We will showcase the usefulness of sarasate by example problems in the context of distributed policy making as discussed in the above.

2 MOTIVATION

<table>
<thead>
<tr>
<th>P₁ dest path</th>
<th>Q(P₁) dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.3.4 x [123]</td>
<td>1.2.3.4 x=123</td>
</tr>
<tr>
<td>5.6.7.8 x [123]</td>
<td>5.6.7.8 x=123</td>
</tr>
</tbody>
</table>

Table 1: Conditional tables also capable of representing answers to queries (on policies).

Conditional tables as policy representation. Viewing a policy by the set of network states it permits, our goal is to find a representation for those states. In this demonstration, we advocate the use of conditional tables. Conditional tables augment regular tables (whose contents are concrete data) with variables and an additional column that holds conditions over the variables. A conditional table, depending on the instantiation of the variables, can correspond to many concrete values. In the many possible concrete instances, a tuple is presented only if the condition holds. To see the strength of conditional tables, consider a form of load balancing policy, represented by P₁, which says path [123] will be used for 5.6.7.8 only if it is not already allocated to 1.2.3.4. Evaluating query Q appending constraint x=123 in the condition column on P₁ produces Q(P₁) which encodes the answer set — {(1.2.3.4), (5.6.7.8)} (either of the prefixes is a correct answer, but not simultaneously) — precisely.

A strong policy representation system. In addition to join and select discussed above, all relational operators on conditional tables can be performed in exactly the same way as in the case of the usual relations. This makes conditional tables a strong representation system for network policies, which we call relational policies, as follows: for a conditional table r in Figure 1, which represents a network policy and maps to (via variable valuations Rep) all possible legitimate states, denoted by a set of regular tables S (= Rep(P)); given a relational query Q, when computing the answers to Q(P₁), denoted by P₁, we can think of some unknown legitimate state s ∈ S — as the current true network state — being queried by Q, producing s=Q(s); This computation is both safe — querying a policy (Q(P₁)) will only return information that does correspond to query on some legitimate state (Q(s)); and complete — querying a policy (Q(P₁)) will return all the information found by querying any legitimate state s ∈ S (Q(s)). In this sense, relational policies lift network policies to first class data objects that can be accessed and processed in exactly the same way as the network states they correspond to.

3 DEMONSTRATION PLAN

We implement sarasate in the PostgreSQL database (the world’s most advanced open source relational database) [21]. This is especially important as it allows us to leverage existing database structure (e.g., indexing) to accelerate evaluation. The challenge is that Postgres (like most databases), does not support c-tables: the existing data fields and SQL operations do not permit c-variables, and the default SQL evaluation cannot be easily altered to account for conditions. Fortunately, we give a straightforward method to bypass the default valuation in three steps: first, we use pure SQL to generate the regular data part of a c-table without conditions with some key terms (constants) reserved for c-variables; next, the conditions manipulation (including pattern matching) is implemented by a sequence of SQL UPDATE, finally, optional the Z3 solver [6] is invoked to remove tuples with contradictory conditions.

Figure 2: User interface of our prototype: executing arbitrary SQL queries over relational policies.

The result is sarasate, a prototype that retains the performance of the Postgres as much as possible. In this demonstration, we will show, as illustrated in Figure 2, how our prototype can handle arbitrary SQL queries over relational policies via synthetic policies. In addition, we will show realistic examples using the BGP ribs and updates from the routview project [25], including simulating BGP updates and exchanging routing policies between neighboring ASes.

ACKNOWLEDGEMENTS

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