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 polices 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 polices allow us to extend a rich set of data mediating methods to address the networking problem of coordinating polices 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.

SIGCOMM '21 Demos and Posters, August 23–27, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8629-6/21/08…$15.00
https://doi.org/10.1145/3472716.3472848

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 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

Bin Gui, Fangping Lan, Anduo Wang
Temple University

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

P_L	dest	path	
	1.2.3.4	x	
	5.6.7.8	[123]	x≠[123]

$Q(P_L)$	dest	
	1.2.3.4	x=[123]
	5.6.7.8	x≠[123]

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 `P_L`, 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 `P_L` produces `Q(P_L)` 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 `P` 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 $\mathcal{S}$ (= *Rep*(*P*)); given a relational query `Q`, when computing the answers to `Q(P)`, denoted by `P'`, we can think of some unknown legitimate state `s`∈ $\mathcal{S}$ — as the current true network state — being queried by `Q`, producing `s'=Q(s)`; This computation is both safe — querying a policy (`Q(P)`) will only return information that does correspond to query on some legitimate state (`Q(s)`); and *complete* — querying a policy (`Q(P)`) will return all the information found by querying any legitimate state `s`∈ $\mathcal{S}$ (`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.

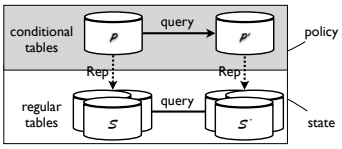

Figure 1: Conditional tables form a strong representation for network policies.

3 DEMONSTRATION PLAN

We implement sarasate in the PostgreSQL database (the worlds'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.

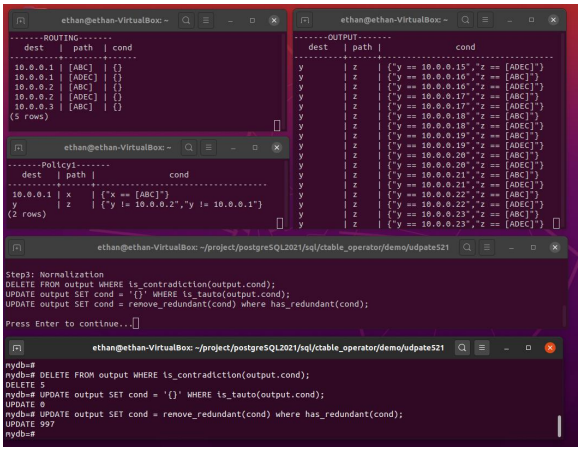

Figure 2: User interface of our prototype: executing arbitrary SQL queries over relational policies.

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 routeview 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.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu (Eds.). 1995. *Foundations of Databases: The Logical Level* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. 1987. On the Representation and Querying of Sets of Possible Worlds. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '87)*. Association for Computing Machinery, New York, NY, USA, 34–48. https://doi.org/10.1145/38713.38724

[3] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 155–168. https://doi.org/10.1145/3098822.3098834

[4] Ryan Beckett, Ratul Mahajan, Todd Millstein, Jitendra Padhye, and David Walker. 2016. Don'T Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations. In *Proceedings of the 2016 ACM SIGCOMM Conference* (Florianopolis, Brazil) *(SIGCOMM '16)*. ACM, New York, NY, USA, 328–341. https://doi.org/10.1145/2934872.2934909

[5] Rudiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, and Martin Vechev. 2020. Config2Spec: Mining Network Specifications from Network Configurations. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 969–984. https://www.usenix.org/conference/nsdi20/presentation/birkner

[6] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (Budapest, Hungary) *(TACAS'08/ETAPS'08)*. Springer-Verlag, Berlin, Heidelberg, 337–340. http://dl.acm.org/citation.cfm?id=1792734.1792766

[7] Sean Donovan and Nick Feamster. 2014. Intentional Network Monitoring: Finding the Needle without Capturing the Haystack. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks* (Los Angeles, CA, USA) *(HotNets-XIII)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/2670518.2673872

[8] Nick Feamster, Hari Balakrishnan, and Jennifer Rexford. 2004. Some foundational problems in Interdomain routing. In *In HotNets, 2004. (Cited on*. 41–46.

[9] Nick Feamster, Jay Borkenhagen, and Jennifer Rexford. 2001. Controlling the Impact of BGP Policy Changes on IP Traffic.

[10] Nate Foster, Arjun Guha, Mark Reitblatt, Alec Story, Michael J. Freedman, Naga Praveen Katta, Christopher Monsanto, Joshua Reich, Jennifer Rexford, Cole Schlesinger, David Walker, and Rob Harrison. 2013. Languages for software-defined networks. *IEEE Communications Magazine* 51, 2 (2013), 128–134. http://dblp.uni-trier.de/db/journals/cm/cm51.html#FosterGRSFKMRRSWH13

[11] Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. 2002. The Stable Paths Problem and Interdomain Routing. *IEEE Trans. on Networking* 10 (2002), 232–243.

[12] T. G. Griffin and G. Wilfong. 2000. A Safe Path Vector Protocol. In *INFOCOM*.

[13] Joel M. Halpern and Carlos Pignataro. 2015. Service Function Chaining (SFC) Architecture. RFC 7665. https://doi.org/10.17487/rfc7665

[14] Tomasz Imieliundefinedski and Witold Lipski. 1984. Incomplete Information in Relational Databases. *J. ACM* 31, 4 (Sept. 1984), 761–791. https://doi.org/10.1145/1634.1886

[15] Xin Jin, Jennifer Gossels, Jennifer Rexford, and David Walker. 2015. CoVisor: A Compositional Hypervisor for Software-defined Networks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (Oakland, CA) *(NSDI'15)*. USENIX Association, Berkeley, CA, USA, 87–101. http://dl.acm.org/citation.cfm?id=2789770.2789777

[16] Hyojoon Kim, Joshua Reich, Arpit Gupta, Muhammad Shahbaz, Nick Feamster, and Russ Clark. 2015. Kinetic: Verifiable Dynamic Network Control. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation* (Oakland, CA) *(NSDI'15)*. USENIX Association, Berkeley, CA, USA, 59–72. http://dl.acm.org/citation.cfm?id=2789770.2789775

[17] Ratul Mahajan, David Wetherall, and Thomas Anderson. 2005. Negotiation-based routing between neighboring ISPs. In *NSDI*.

[18] Ratul Mahajan, David Wetherall, and Thomas Anderson. 2007. Mutually controlled routing with independent ISPs. In *NSDI*.

[19] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (March 2008), 69–74. https://doi.org/10.1145/1355734.1355746

[20] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. 2013. Composing Software-defined Networks. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation* (Lombard, IL) *(nsdi'13)*. USENIX Association, Berkeley, CA, USA, 1–14. http://dl.acm.org/citation.cfm?id=2482626.2482629

[21] PostgreSQL: The World's Most Advanced Open Source Relational Database. [n.d.]. https://www.postgresql.org/.

[22] Chaithan Prakash, Jeongkeun Lee, Yoshio Turner, Joon-Myung Kang, Aditya Akella, Sujata Banerjee, Charles Clark, Yadi Ma, Puneet Sharma, and Ying Zhang. [n.d.]. PGA: Using Graphs to Express and Automatically Reconcile Network Policies. In *SIGCOMM '15*.

[23] Joshua Reich, Christopher Monsanto, Nate Foster, Jennifer Rexford, and David Walker. 2013. Modular SDN Programming with Pyretic. *USENIX ;login* 38, 5 (October 2013).

[24] Y. Rekhter, T Li, and S Hares. 2006. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. RFC Editor. http://www.rfc-editor.org/rfc/rfc4271.txt

[25] Route Views. [n.d.]. Route Views. http://www.routeviews.org/routeviews/.

[26] Thomas Wirtgen, Quentin De Coninck, Randy Bush, Laurent Vanbever, and Olivier Bonaventure. 2020. XBGP: When You Can't Wait for the IETF and Vendors. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks* (Virtual Event, USA) *(HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3422604.3425952

[27] Wen Xu and Jennifer Rexford. 2006. MIRO: Multi-path interdomain routing. In *ACM SIGCOMM*.

[28] Yifei Yuan, Dong Lin, Ankit Mishra, Sajal Marwaha, Rajeev Alur, and Boon Thau Loo. 2017. Quantitative Network Monitoring with NetQRE. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 99–112. https://doi.org/10.1145/3098822.3098830

[29] Peng Zhang, Yuhao Huang, Aaron Gember-Jacobson, Wenbo Shi, Xu Liu, Hongkun Yang, and Zhiqiang Zuo. 2020. Incremental Network Configuration Verification. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks* (Virtual Event, USA) *(HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 81–87. https://doi.org/10.1145/3422604.3425936