Rethinking Network Policy Coordination: A Database Perspective

Anduo Wang
Temple University
adw@temple.edu

Seungwon Shin
KAIST
claude@kaist.ac.kr

Eduard Dragut
Temple University
edragut@temple.edu

ABSTRACT

Database usage in the context of networking has been focusing on managing factual data — network state. But database systems are also renowned for mediating among *semantic data* — data integrity constraints (ICs) that capture network policies. This paper asks if and how can database systems help with coordinating network policies in the semantically rich environment of SDN and BGP. We identify several problems — disparate policies buried in the network that hinders rather than facilitates coordination; manual control flow orchestration of SDN policies that burdens the SDN programmer; and overlooked conflicts among interdomain routing policies that, though induced by multiple ASes, are only manifested within a single AS. Driven by these unique problems, we present a preliminary database solution that, using ICs as a unifying knowledge representation, employs automated reasoning to anticipate and to adjust the interplay between policies and the rest of the networking world.

CCS CONCEPTS

• **Networks** → **Network protocol design**; *Programming interfaces*; *Network manageability*.

KEYWORDS

knowledge representation, BGP, SDN, residue method

ACM Reference Format:
Anduo Wang, Seungwon Shin, and Eduard Dragut. 2019. Rethinking Network Policy Coordination: A Database Perspective. In *APNet '19: 3rd Asia-Pacific Workshop on Networking, August 17–18, 2019, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3343180.3343193

APNet '19, August 17–18, 2019, Beijing, China

ACM ISBN 978-1-4503-7635-8/19/08…$15.00
https://doi.org/10.1145/3343180.3343193

1 INTRODUCTION

There have been increased interests in applying database concepts and techniques to modern networking. Data management languages were adapted for distributed protocols in declarative networking [15]. In the context of software-defined networking (SDN), database transaction processing was used for synchronizing network states [4, 12, 14]. More recently, we take the entire network under a relational database [22] and use the high-performance database system as *the* controller. These efforts demonstrated the many benefits of using database systems for managing network state — the factual data. But database systems are also renowned for mediating among semantic data — data integrity constraints about what are the acceptable data.

In this paper, we ask if and how can database systems help with managing network policies that can interact in complex ways. As a first step, we reexamine policy interactions in the semantically rich environment of SDNs and interdomain routing. We identified several problems that are either overlooked or hard to manage in the current network infrastructure, but may benefit from an alternative database approach.

First, network policies buried in the network can take drastically different forms. Policies can be embedded in the configurations of network devices such as a switch or a firewall, wired in disparate extensions to the border gateway protocol (BGP), the de-facto protocol and carrier for Internet policies [20], or coded in a control software running in SDNs [11, 19]. Such low level heterogeneous representation hinders rather than facilitates coordination. In contrast, database systems use integrity constraints (ICs) as a unified knowledge representation that can lift the policies out of the network, opening up the door to automated coordination.

Second, in a centralized environment like SDNs, polices are often intended for distinctive aspects of the network. Such disjoint polices are jointly satisfiable and, collectively, oversee the entire life of a single task. Yet, they are not necessarily independent. A network operation meant to maintain one policy might inadvertently violate another. The popular SDN solution is modular programming of the controller software in which the polices live, and leaves the onus of forming a correct modular composition to the programmer, a job that only gets worse as the controller grows. In contrast, a database system might achieve the same coordination

by mediating the responsible network operations — a data update problem that also takes into account the polices.

Finally, in the decentralized Internet, a network of interconnected autonomous systems (ASes), policies set by distributed ASes to achieve their potentially competing goals can conflict. Despite extensive studies on route oscillation — an anomaly across AS borders — and various seminal remedies [6, 10], there still exist overlooked conflicts. In particular, we made a case for conflicts that manifest themselves within an AS: With BGP and its many extensions, an AS often attempts to control path selection in a remote neighbor. That neighbor, when being a common target by several such influential ASes — unaware of and with no visibility into each other, might has to deal with conflicting effects of those ASes. In contrast, the effects of ICs are well understood in database systems, they are routinely processed with all relevant knowledge extracted. We argue that we can leverage similar knowledge processing to predicate the interplay among several AS policies, and to adjust them accordingly.

As a realization of these ideas, we present a preliminary database solution, as a means towards a policy system for modern networks that is more predictable, flexible, and manageable. Specifically, we make the following contributions:

- *Unifying knowledge representation.* We lift the disparate policies out of the network by introducing a unifying knowledge representation with data integrity constraints (ICs) [7]. ICs are logical statements about what are acceptable network state, offering a firm ground for automated policy coordination.
- *Managing complementary policies.* We develop an update orchestrater that coordinates complementary SDN policies by ordering the corresponding operations. To this end, we introduce semantic dependency to capture whether an update, meant to restore one policy but introduces new policy violations, requires additional updates to maintain the overall policy compliance.
- *Managing conflicting policies.* We develop a policy negotiator that enables an AS to anticipate conflicts among "external" policies exerted by neighboring ASes, and to adjust policies in conflict by automated transformation. To this end, we extend the knowledge reasoning techniques used in semantic query processing to IC processing.

2 MOTIVATING SCENARIOS

We examine complementary policies that oversee a single network task in SDN and make a case for conflicts among interdomain routing policies that are manifested within a single AS. In both cases, we discuss the challenges and opportunities, and highlight the benefits a database solution may bring.

Complementary policies

Network policies in a centralized environment are often deliberately set for distinct objectives that are jointly satisfiable. These policies do not conflict, but are not independent of each other either. Instead, they complement each other: The disjoint policies oversee the entire life of a single task; The network operations generated to realize these policies, when performed in a proper order, complete that task.

The policies live in the SDN controllers — network invariants maintained by the control software — fall into this category. SDN programmers often create complex SDN tasks from multiple simpler software modules, each realizes a particular simple policy. These modules continuously monitor and reconfigure the network — checking for policy violation and generating repairing updates. For example, secure routing can be broken into a security module and a routing module. The security module filters out suspicious traffic requests — maintaining a policy that no traffic in a blacklist can enter, the routing module configures data-plane path — maintaining a policy that every path must match a traffic request. Here, proper coordination relies on a strict ordering: the security update (checking new traffic request against the blacklist) must always precede the routing update (path installation).

To achieve such coordination, the current approach is modular programming. Though many SDN programming platforms [17, 19] exist, the onus of forming a correct coordination logic still falls on the programmer. The programmer has to clearly understand every policy implicitly embedded in the software, their joint intent, and to anticipate a comprehensive control flow for all possible interactions. Besides, the coordination logic, being also embedded in the software such as a master program is hard to maintain or extend [17]: as the SDN evolves with new modules, existing ones might need to be decomposed and re-composed.

In contrast to the manual control flow coordination required in modular programming, we seek a data centric alternative. If we view the entire SDN state as data, the network operations as data updates, we ask if we can automatically coordinate the SDN software by controlling the responsible data updates? Database systems are renowned for mediating multiple data updates via concurrency control of the reads and writes. But concurrency control only achieves data integrity among independent updates. Can we develop a new update orchestrater to bring policy compliance among interacting updates? If successful, such a data solution can free the SDN programmers from the daunting job of modular programming, and facilitate the evolution of SDNs beyond their initial deployments.

Conflicting policies

Network policies in a decentralized environment run into conflicts when a network object compliant with one policy is considered a violation by another. A well-studied example is the stable path problem in interdomain routing [5, 9]: multiple path preferences for a common destination, set by distributed ASes with potentially competing goals, can disagree and fail to converge to a global path decision, leading to an anomaly called permanent route oscillation. Here, the root cause is the contradicting choices made about a common object by multiple policy makers — i.e. ASes. *But, does it follow that the manifestation of a conflict must also involve multiple ASes — like in route oscillation?*

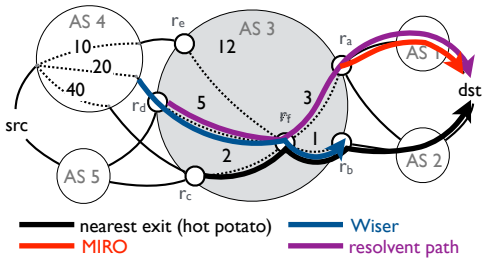

Figure 1: Path selection (to dst) within AS3 is jointly determined by the local nearest exit policy of AS3, the traffic engineering policy of AS4, and the security policy of AS5.

Specifically, assuming each AS makes consistent route decision within its boundary, can we rule out policy conflicts happening within an AS? We refute this seemingly true statement by the following example. Consider route selection for dst in AS3, shown in Figure 1. AS3 has a local nearest exit policy — the numbers depict the IGP costs — that picks a path with lowest cost (highlighted in bold black). But AS3 is also influenced by its upstream neighbors AS4, AS5: AS5 considers AS2 insecure which, expressed as a MIRO policy [24] — a BGP extension that enables negotiating alternative paths, drives AS3 to avoid AS2 in the downstream (in red); AS4, on the other hand, being concerned about traffic engineering, runs the Wiser extension [16] with AS3 to select a path that jointly optimizes cost in AS3 and itself (in blue). These policies conflict as they disagree about what path AS3 should use — black (local policy with IGP), red (security with MIRO), or blue (TE with Wiser)?

More generally, path selection within an AS is *not isolated* from the external world — its neighbors [16, 18, 24]. Prior work on route oscillation [6, 10] does not address this type of conflicts. In fact, those solutions often assume monolithic policies within individual ASes. In addition, while our conflicting example is admittedly contrived, we envision that, several trends in networking research and practice — more flexible policies [8, 16, 24] and architectural support for multiple policy extensions to coexist [21] — can only fuel such conflicts.

We ask how to find a path choice (highlighted in purple) that reasonable matches all three policies? If we model network policies by ICs, can we develop new techniques similar to database semantic transformation — transformation of a query by knowledge extracted from an IC — to anticipate and resolve the impact of a policy? ICs in a database system is routinely processed to facilitate query processing, resulting in transformed query that gives equivalent answer but evaluates faster [2, 7]. But such IC techniques is designed for an IC and a query, can we develop a new knowledge reasoning technique to infer and control the interplay among multiple network policy ICs? If successful, such a database solution can help evolve the Internet towards a policy system that is more flexible, easier to understand and manage.

3 A DATABASE SOLUTION

To answer the questions raised in § 2, this section presents a preliminary database solution. Our observation is that database system not only manages factual data — relations that are suitable for representing network state, it also mediates among semantic data — ICs that naturally express network policies. Modeling network policies by ICs — logical statement about what are preferable network state — unifies various network policies that are previously buried in different parts of the network. More importantly, the logic-based ICs lend themselves to automated reasoning, enabling the following coordination services:

A new update orchestrater for complementary policies that predicates how a network update affects a policy IC. The orchestrater orders the updates in such a manner that, any policies that have been previously restored by some update will never be violated by updates scheduled afterwards.

Policy negotiator for conflicting policies that jointly determine a single network. The negotiator predicates the interplay between two policy ICs, and uses knowledge from one policy IC to transform another policy, in such a manner that a less important IC will be semantically constrained to ensure compliance with the more important one.

3.1 A Knowledge Representation

We view network policies as *declarative properties of the network states that constrain* — rather than *operational instructions that generate* — the network behaviors.

Network state as relations

First, we review the database representation of the entire network state proposed in our prior work [22]. A network state — traffic conditions, topologies, configurations etc— is a finite set of ground relations (predicates), implemented by database tables and views. To simplify discussion in the rest

of the paper, we assume a data schema as follows. We note that the particular choice of the scheme is insignificant.

```
% intradomain tables
tp(sid,nid)          % topology
rm(fid,sid,nid)      % end-to-end reachability
    (matrix)
cf(fid,sid,nid)      % configuration (forwarding
    table)
path(pv,cost,iid,eid)% internal path
% interdomain tables
aspath(did,rid,apv)  % AS level path
```

Among the intradomain tables, `tp` is the internal router-level topology table that stores link pairs `(sid,nid)`, `rm` is the end-to-end reachability matrix table that keeps flow requirement identified by `(fid)` between two hosts `sid,nid`, `cf` is the router configuration that describes for each flow `(fid)` the next-hop configuration: for the current router `sid`, the next hop is `nid`. `path` table specifies for every path `pv` (path vector) the internal cost `cost` between the ingress router `iid` and the egress router `eid`. The interdomain table structures network state exchanged across domain borders: `aspath(did,eid,apv)` represents the AS-level paths to destination `did` through the border router `eid` along an AS level path vector `apv` (list of AS identifiers along the path).

Each tuple (i.e. data item) in the relation is a ground (free of variables) predicate that encodes a fact — factual knowledge of the network state. For example, `tp(X,Y)` is asserts a link between two nodes `X,Y`.

Network policy as integrity constraint (IC)

A network policy is collection of logical statements over the network predicates, called ICs, stating what constitutes an acceptable state. We adopt the *denial rule* of the form $:-b_1,\ldots,b_n$, meaning that the body predicates b_i, when jointly satisfied, will lead to `False` (empty head). A denial rule eliminates any network state simultaneously satisfies all the body predicates. For example, we represent the routing and security policies in § 2 as follows:

```
% routing policy
IC1 :- ¬rm(F,S,D),cf(F,X,Y).
IC2 :- rm(F,S,D),¬cf(F,X,Y).
% security policy
IC3 :- rm(F,S,D),blacklist(S,D)
```

The routing policy is referential constraints that require the per-switch configuration (`cf`) to match traffic requests (`rm`): IC_1 forbids a switch configuration `cf(F,X,Y)` that does not correspond to any traffic request `¬rm(F,S,D)`, IC_2 avoids missing switch configuration for an existing traffic request. The security IC_3 prevents the network from installing any path for traffic that also appears in a blacklist. Similarly, the MIRO interdomain policy that prevents the Internet from selecting a path through `AS2` can be expressed by IC_4.

```
% MIRO policy
IC4 :- aspath(D,N,P), ('AS2' in P).
```

To see the strength of the IC representation, consider the Wiser policy of `AS3` that jointly optimize end to end path from `AS4` to `AS3`.

```
% Wiser policy
Wiser(D,Rp,C') :-
    aspath(D,Re,P),path(P',C,Rp,Re),C'=norm(C).
best(D,min<C>) :-
    Wiser(D,Rp,C1),Advertise(Rp,C2),C=C1+C2.
IC5 :- Wiser(D,Rp,C),best(D,C'),C>C'.
```

`Wiser` is a view created by `AS3` that computes — with the help of the auxiliary function `norm` — a normalized path cost by joining `aspath` and `path` of `AS4`. Suppose `AS4` creates and exposes to `AS3` a view `Advertise(R,C)`, advertising the normalized path cost `C2` for the internal path through the peering router `Rp`. With `Advertise`, `AS3` can express its Wiser policy as a denial rule that excludes from consideration any peering router `Rp` that leads to a path with a higher cost (`C>C'`).

3.2 Managing Complementary Polices

This section presents an update orchestrater that allows an SDN to be controlled by multiple control polices without worrying about how to form a coherent coordination logic.

Essentially, we introduce *semantic dependency* to organize updates into a hierarchy called *semantic layering*. Updates at a higher layer can trigger future updates to the lower layer, as the upper layer update introduces new violations to polices maintained by the lower layers. That is, the upper layer update is complete — finally transform the network into a new state compliant with all polices — only when all the updates at lower layers are also complete.

Semantic dependency

Suppose two polices X and Y maintain IC_X and IC_Y via updates U_X and U_Y, respectively. U_X semantically depends on U_Y if (1) U_X can introduce new violations to IC_X, i.e. if U_X can render IC_Y False on some network state; (2) but the updates of Y can never violate X policy, i.e.U_Y can never falsify IC_X. Given a set of polices and all the update dependencies, a topological sort over the dependency graph will give an ordering among the updates.

As an example of semantic dependency, consider the security and routing polices (§ 2), suppose the update restoring security, denoted by U_{sec}, is to drop any unsafe traffic requests — delete `rm` entries that also appear in some `blacklist` table; The update for maintaining routing, denoted by U_{rt}, is to add (resp. remove) a path — `cf` entries — for traffic requests (resp. withdraws) in `rm`. U_{sec} will introduce transient violation to

IC_2 until the routing application takes action to remove the corresponding configurations in `cf`. But U_{rt} will never affect security policy. Thus U_{sec} depends on U_{rt}.

Determine semantic dependency

We can automatically determine semantic dependency by database irrelevance reasoning [1]. The trick is to cast semantic dependency as an irrelevant update problem: we associate with each IC `:-` $b_1 \ldots b_k$ an auxiliary view v_{IC}`(atts):-` $b_1 \ldots b_k$. v_{IC} materializes violation to the IC, also called violation view, where `atts` are the variables in the body of the IC (i.e.$b_1 \ldots b_k$).

With the violation view v_{IC} — a virtual table computed by a query, semantic dependency is reduced to checking whether a database update can alter the result of v_{IC} (query). In other words, X depends on Y if U_X is relevant to (can alter) v_{ICy} but U_Y is irrelevant to v_{ICx}. The database community developed a method to automatically check (ir)relevance by satisfiability reasoning [1].

We show the intuition of (ir)relevant update reasoning by an example. Consider again the security and routing policy. The routing update cannot alter security view, because either deletion nor insertion on `cf` touch the body of security view, namely `rm` or `blacklist`. The security update, however, is the deletion of suspicious traffic, characterized by the condition `rm(F,S,D)/\blacklist(S,D)`. But this condition and the routing view body is jointly satisfiable. Thus there exists some security deletion that can alter the routing view.

3.3 Managing Conflicting Policies

This section presents a policy negotiator that allows an AS to be jointly controlled by policies set by multiple (potentially competing) neighbors.

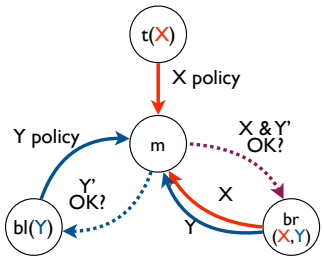

Figure 2: Policy negotiation

Figure 2 shows the typical operation of the negotiator deployed at AS `AS m`: Suppose `AS t` (top) proposes to `AS m` a policy `X`, `AS bl` (bottom left) proposes a policy `Y`, but `X` conflicts with `Y`; Or `AS br` (bottom right) proposes two policies `X,Y` without being aware of the conflicts between them at `AS m`. In both cases, the policy negotiator will automatically detect conflicts and recommends a resolution — if `AS m` favors `X` over `Y`, the policy negotiator will transform `Y` into `Y'` that takes into account `X`, and negotiate the `X` compliant `Y'` with `AS bl`, or the `X&Y'` with `AS br`.

Central to the policy negotiator is a policy transformation process based on the residue method. The residue method is a semantic transformation process, it was originally developed for query optimization where ICs were processed to semantically constrain a query to accelerate evaluation. The idea is that, the impact — all useful information — of a policy can be extracted by a theorem proving technique called (partial) subsumption [3]. Given two policies `X` and `Y` as inputs, partial subsumption outputs a fragment of `X` called *residue* that characterizes the additional conditions that must be satisfied for `Y` to be compliant with `X`. That is, residue anticipates the impact of `X` on `Y`.

(Partial) subsumption. For two policy ICs `P,Q`, `P` subsumes `Q` — `P` is more general — if there exists a substitution that makes `P` a subclause of `Q`. Take IC_M and $IC_{M'}$ (in the following code) as an example: IC_M is a policy that requires all paths to bypass `AS2`, $IC_{M'}$ is a more specific policy that avoids `AS2` only for paths to a malicious destination. IC_M subsumes $IC_{M'}$ as policy `M` entails the more specific `M'`. That is, partial subsumption implies compliance: any network consistent with `M` is also compliant with `M'`.

```
1 IC_M  :- aspath(I,R,P), ('AS2' in P).
2 IC_M' :- aspath(D,X,Y), ('AS2' in Y), D='malicious_d'.
3 IC_d  :- aspath(D,X,Y), D='malicious_d'.
```

What interests us is *partial* subsumption that indicates policy conflicts. IC `Q` is partially subsumed by IC `P` when `Q` is subsumed by a *subclause* of `P`. For example, IC_M partially subsumes — via the subclause `:- aspath(I,R,P)` — IC_d, an alternative security IC that simply avoids the malicious destination. These two policies conflict — a path avoiding `malicious_d` but waypointing `AS2` is considered safe by IC_d but perceived insecure by IC_M. Similarly, one can find a path that agrees with IC_M but violates IC_d.

Detecting policy conflicts by residue. To detect conflicts, we only need to check for partial subsumption by leveraging the subsumption algorithm developed in [3]. The subsumption algorithm tries to construct a refutation tree with the subsuming clause as the root and elements from the subsumed clause for resolution. The fragment left at the bottom of the refutation tree is called *residue* — it states a condition that must be satisfied for the to-be-subsumed clause to be compliant with the subsuming clause. A "null" residue forms a proof of subsumption, whereas a non-trivial residue — neither "null" nor redundant — explains what is still lacking that lead to conflicts.

For example, the refutation tree constructed for IC_M and $IC_{M'}$ in Figure 3 (left) gives a "null" residue, thus proves `M'`'s compliance with `X`. On the other hand, the refutation tree in Figure 3 (right) gives a non-trivial residue `:- ('AS2' in P)`, meaning that an extra condition — `¬('AS2' in P)` — must be

taken into account for the alternative security policy to be compliant with M.

Resolving policy conflicts by residue annotation. Residue also explains conflict, characterizes what is missing, thus offers a natural fix. We only need to add the residue to the policy that is to be transformed for conflict resolution. Specifically, for two conflicting policies P,Q where P is considered more important, the policy negotiator operates as follows: It runs the subsumption algorithm that takes P as the subsuming clauses, and outputs a non-trivial residue r. The residue r is then included in Q to remove the conflict. For example, to annotate the residue in Figure 3 (right), we first add it to the body of IC_d. Then, the residue is interpreted as an additional logical condition.

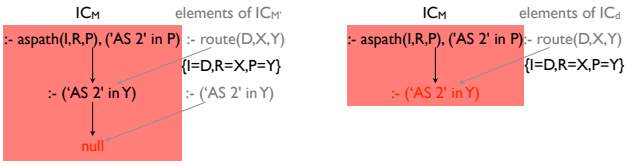

Figure 3: Refutation tree: the residue (in red) at the bottom shows the impact of MIRO policy (IC_M) on the more specific MIRO policy ($IC_{M'}$ on the left) and the security policy (IC_d on the right)

```
% security IC annotated with MIRO residue
:- aspath(D,X,Y), D='malicious_d', {:- 'AS2' in Y}.
% MIRO compliant security policy (after unfolding
    the residue)
:- aspath(D,X,Y), D='malicious_d', ¬('AS2' in Y).
```

4 RELATED WORK

Policy languages. SDN languages offer various constructs for expressing policies as an integrated part of the control software. Declarative networking [12, 15] uses a language construct closer to our knowledge representation. However, policies are still embedded in the program. In contrast, our knowledge representation decouples the policies from the control plane, and enables automated reasoning for a broader set of policy interactions.

SDN policy coordination. Many SDN platforms use modular programming to facilitate policy composition — e.g., Pyretic [19] introduces specialized composition operators to stitch together policy functions, CoVisor [13] uses OpenFlow rule composition as the common interface to combine policies developed in different platforms. What tells us apart is that we aim for system level support that automates composition. In interdomain routing, restricted policy languages and policy guidelines were proposed to ensure global path convergence among multiple AS path preferences. Our work addresses a different overlooked form of conflicts that occur within an AS.

BGP policy coordination. Many extensions to BGP [8, 16, 23, 24] attempted to enable more flexible BGP policies. These proposals often consider backward compatibility with the legacy BGP system but provide very little support for joint control among themselves. The only work we are aware of that considers the co-existence of many BGP variants is D-BGP [21] which studied architectural features needed for multiple interdomain protocols that are partially deployed. But D-BGP still assumes a single extension within an AS. To our best knowledge, our work is the first attempt to allow the AS to simultaneously deploy multiple extensions.

5 LIMITATIONS

This section acknowledges several limitations and discusses possible mitigation plans. First, the IC representation is unconventional and requires some effort to express common policies like path ranking. We plan to perform a comprehensive expressiveness study, and compile a template for common SDN and BGP policies. An alternative is to develop supporting tools to bridge popular policy configurations and the ICs. Second, using ordering of network updates as a means to coordinate SDN policies requires the update dependency to be acyclic. We consider cyclic dependency a very interesting and practical problem, but more theoretical studies are needed. Nevertheless, for acyclic policies, our method still improves over existing solutions. Finally, ASes in the interdomain may not share their policies due to commercial concerns. As ongoing work, we are policy anonymization technique with inductive generalization. Instead of exchanging plain policy ICs, ASes can advertise an aggregate by computing their inductive generalization. That being said, a comprehensive solution to interdomain policy exposure may be impossible, and is beyond the scope of this work.

6 CONCLUSION

We reexamined several challenges in coordinating policies of modern networks: the disparate policies buried in different parts of the network that hinder rather than facilitates coordination; the complementary SDN policies that, to cooperate to complete a single task, can interact in subtle ways as the network updates meant to restore one policy inadvertently violate another; and the overlooked conflicts within an AS caused by multiple neighbors. We discuss why existing solutions fall short and present an alternative database approach. Our solution explores the data integrity constraints as a unifying knowledge representation for network policies, and employs logic reasoning about ICs to precisely and automatically predicate and adjust the behavior of the network policies.

REFERENCES

[1] J. A. Blakeley, N. Coburn, and P.-V. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Trans. Database Syst.*, 1989.

[2] U. S. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.*, 15(2):162–207, June 1990.

[3] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, Inc., 1997.

[4] B. Davie, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. Gude, A. Padmanabhan, T. Petty, K. Duda, and A. Chanda. A database approach to sdn control plane design. *SIGCOMM Comput. Commun. Rev.*, 47(1):15–26, Jan. 2017.

[5] N. Feamster, R. Johari, and H. Balakrishnan. Implications of autonomy for the expressiveness of policy routing. *IEEE/ACM Trans. Netw.*, 15(6):1266–1279, Dec. 2007.

[6] L. Gao and J. Rexford. Stable Internet routing without global coordination. In *ACM SIGMETRICS*, 2000.

[7] P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity constraints: Semantics and applications. In *Logics for Databases and Information Systems*, 1998.

[8] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *ACM SIGCOMM*, 2009.

[9] T. G. Griffin, A. D. Jaggard, and V. Ramachandran. Design principles of policy languages for path vector protocols. SIGCOMM '03, pages 61–72, New York, NY, USA, 2003. ACM.

[10] T. G. Griffin and J. L. Sobrinho. Metarouting. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, pages 1–12, New York, NY, USA, 2005. ACM.

[11] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[12] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker. Fml: Practical declarative network management. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, pages 1–10, New York, NY, USA, 2009. ACM.

[13] X. Jin, J. Gossels, J. Rexford, and D. Walker. Covisor: A compositional hypervisor for software-defined networks. NSDI'15, pages 87–101, Berkeley, CA, USA, 2015. USENIX Association.

[14] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker. Onix: a distributed control platform for large-scale production networks. OSDI'10, 2010.

[15] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative routing: Extensible routing with declarative queries. SIGCOMM '05. ACM, 2005.

[16] R. Mahajan, D. Wetherall, and T. Anderson. Mutually controlled routing with independent ISPs. In *NSDI*, 2007.

[17] C. Prakash, J. Lee, Y. Turner, J.-M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang. Pga: Using graphs to express and automatically reconcile network policies. In *SIGCOMM '15*.

[18] B. Quoitin, C. Pelsser, L. Swinnen, O. Bonaventure, and S. Uhlig. Interdomain traffic engineering with bgp. *Comm. Mag.*, 41(5):122–128, May 2003.

[19] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker. Modular sdn programming with pyretic.

[20] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271, RFC Editor, 2006.

[21] R. R. Sambasivan, D. Tran-Lam, A. Akella, and P. Steenkiste. Bootstrapping evolvability for inter-domain routing with d-bgp. SIGCOMM '17, New York, NY, USA, 2017. ACM.

[22] A. Wang, X. Mei, J. Croft, M. Caesar, and B. Godfrey. Ravel: A database-defined network. SOSR '16. ACM, 2016.

[23] Y. Wang, I. Avramopoulos, and J. Rexford. Design for configurability: Rethinking interdomain routing policies from the ground up. *IEEE J.Sel. A. Commun.*, Apr. 2009.

[24] W. Xu and J. Rexford. MIRO: Multi-path interdomain routing. In *ACM SIGCOMM*, 2006.