Design and Implementation of a Strong Representation System for Network Policies

Fangping Lan*, Sanchari Biswas†, Bin Gui‡, Jie Wu§ and Anduo Wang¶
Department of Computer and Information Sciences, Temple University
Philadelphia, PA, USA
Email: {*fangping.lan, †sanchari.biswas, §jie.wu0001}@temple.edu, {‡bguiethan, ¶anduo.wang}@gmail.com

***Abstract*—*Policy* information in computer networking today, such as reachability objectives of a controller program running on a Software Defined Network (henceforth referred to as SDN) or Border Gateway Protocol (henceforth referred to as BGP) configurations independently set by autonomous networks, are hard to manage. This is in sharp contrast to the relational data structured in a database that allows easy access. This paper asks why cannot (or how can) we turn network policies into relational data. One difficulty to such an approach is that a policy does not always translate to a *definite* network snapshot, but rather is fully described only when we include all the possible network states it admits. We propose relational policies that, while capable of representing and manipulating sets of network states in exactly the same way as a single one, form a strong representation system and accurately capture the information in a policy with the usual Structured Query Language (henceforth referred to as SQL) interface. We demonstrate how, like relational database improves application productivity and enables rapid innovation, relational policies allow us to extend the elegant solutions that the database community developed, to mediate multiple data sources in order to address long-standing challenges and new opportunities for autonomous policy making in the distributed networking environment. We also show the feasibility of relational policies by evaluation on synthetic policies and realistic network topologies.**

***Index Terms*—Network Policies, Relational Algebra, Conditional Tables, Network Manageability**

I. Introduction

Policies being a fundamental part of computer networking, many tools have been developed to exploit and/or manage them throughout a network's entire life cycle. Higher level programming abstractions help operators realize reachability objectives in SDNs [1]–[7]; intention-aware monitoring systems leverage network-wide queries to improve monitoring in programmable hardware [8], [9]; BGP configurations [10]–[14] are still the main vehicle affecting routing whenever rich semantics is involved — whether in the global Internet or datacenters; verification tools [15], [16] with varying capabilities, such as expressiveness, scalability, speed, check whether the network configurations, the SDN programs etc actually obey the properties in the formal specification; and synthesizers [13], [17] attempt to convert network specification of varying forms, such as logical assertions, templates, etc, 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 a prefixed mental model makes it more difficult for anyone not involved in writing the controller program to make sense of or debug a policy. 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 sharp contrast to data management in relational database [18] which is remarkably *easier*: The simple self-explanatory relational data model has replaced many non-relational application-specific data structures and gives a common understanding of the data, thus allowing for communication across tasks and between users. The relational database system 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, 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 — mechanisms, dynamics, additional models etc— to fully express a policy, we argue that, like a predicate in set theory is accurately described by the set it corresponds to, it is probably adequate to represent a policy by *all the possible network states* it admits. Based on this idea, we propose relational policies, a system 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 [18]–[20], which extend regular tables with variables and constraints over those variables, operated via an interface that is both intuitive (the familiar SQL operations), and rigorous (safe operations, deriving information only if in legitimate network state) and complete (capturing all the legitimate states).

II. Overview

Relational policies, like relational databases, seek to provide a versatile shared policy component capable of rapid innovations, with the added benefit of exploiting a wealth of solutions already developed in relational database. To demonstrate this, we investigate the mismatch between the assumption most policy handling today rely on, i.e. a prior consistency and a single exclusive enforcement point, and the networking reality that decision making is autonomous in a distributed environment. Specifically, we use relational policies to study two concrete problems: (1) although local policy making can approximate global optimal by participating in some form of information exchange [21]–[26], the exchange is often low-level and ad hoc, and (2) despite most policy-rich tools enforcing, checking, synthesizing a set of coherent policies [15], [16], they say little on how to obtain a consistent policy from disparate sources (e.g.,. teams overseeing overlapping aspects or interacting with parts of the network). We leverage the elegant solutions that the database community developed to mediate multiple (regular) data sources among multiple users [13], [27]–[35]: we give formal and novel extensions of such classic notions as data exchange and schema mapping (on regular tables) into relational policies (conditional tables), and are able to implement these extensions with moderate effort.

Finally, we study the feasibility of relational policies by evaluating our prototype implementation on synthetic policies and realistic network topologies. Network policy coordination offers an interesting application area, and our preliminary implementation might encourage more investigation into this area.

III. Background and Motivation

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. Consider routing policy for two prefixes 1.2.3.4 and 5.6.7.8 over two alternate paths [ABC], [ADC]: suppose if any node owns both the prefixes, traffic bound to them can take any of the two paths. Such uncertain information can still be accurately expressed in tables if we allow variables in the tuples. One such "variable table" P_R is shown in Table I, it has a desirable property: it can handle join smoothly. If we know both alternate paths are safe, as described by table P_S, we can safely join ($\bowtie$) the two to derive that both destinations are safe as shown in the rightmost table.

P_R	dest	path
	1.2.3.4	x
	5.6.7.8	y

P_S	path	security
	x	safe
	y	safe

dest	path	security
1.2.3.4	x	safe
5.6.7.8	y	safe

TABLE I: Variable tables capable of expressing policy (uncertain network states).

However, when we attempt to query this policy table, we run into a problem: which destination uses path [ABC], expressed by (in SQL format) Q:$\pi_{dest}(\sigma_{path=[ABC]})$? This is a simple conditional query, depending on the values taken by the variables, any of the three possible answers — $\{\langle 1.2.3.4\rangle\}$,$\{\langle 5.6.7.8\rangle\}$,$\{\langle 1.2.3.4\rangle$,$\langle 5.6.7.8\rangle\}$ — can be correct, but no variable table can represent all of them. Conditional tables address this limitation by adding an additional column that holds conditions over the variables, which determines

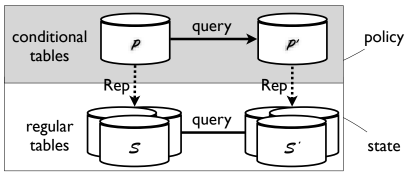

Fig. 1: Conditional tables form a strong representation for network policies.

when a tuple is actually presented. For example, Q(P_R) in Table II is the resulting conditional table that accurately describes all of the correct answers. Note that it can also be interpreted as a new policy that allocates paths through [ABC], like a regular SQL query also represents a regular table containing certain query answers.

Q(P_R)	dest	
	1.2.3.4	x=[ABC]
	5.6.7.8	y=[ABC]

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

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

TABLE II: Conditional tables also capable of representing answers to queries (on policies).

To better demonstrate the strength of conditional tables, we consider a form of load balancing policy, represented by P_L, which says path [ABC] 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=[ABC] in the condition column) on P_L produces Q(P_L) which encodes the answer set — $\{\langle 1.2.3.4\rangle$,$\langle 5.6.7.8\rangle\}$ (either of the prefixes is a correct answer, but not simultaneously) — precisely.

A Strong Policy Representation System. In addition to join and select as discussed above, all relational operators[1] on conditional tables can be performed in exactly the same way as in the case of the usual relations[2]. This makes conditional tables a *strong* representation system for network policies, which we call relational policies. 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); Querying a policy (Q(P)) will only return information that corresponds to the query on some legitimate state (Q(s)) (safe); and all the information found by querying any legitimate state s∈ $\mathcal{S}$ (Q(s)) (complete). 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.

IV. Relational Policy

A. A Strong Representation System with Conditional Tables

The classic notion of a conditional table resembles a regular table but adds two extensions to the data contents: (1)

[1]`SELECT, JOIN, PROJECT, UNION, DIFF` $(\sigma, \bowtie, \pi, \cup, -)$
[2]Formal proof on conditional tables is presented in [18], [20].

P_1	dest	path	
	1.2.3.4	x	x = [ABC]
	y	z	y $\neq$ 1.2.3.5 $\wedge$ y $\neq$ 1.2.3.4

R	dest	path
	1.2.3.4	[ABC]
	1.2.3.4	[ADEC]
	1.2.3.5	[ABE]
	1.2.3.6	[ABE]

$R \bowtie P_1$	dest	path
	1.2.3.4	[ABC]
	1.2.3.6	[ABE]

TABLE III: P_1: a policy that requires static route [ABC] for 1.2.3.4 and filters 1.2.3.5; R: (fragment of) current routing state; $R \bowtie P_1$: routing state after applying the policy.

allowing variables and (2) adding an extra condition column of constraints over the variables, where each tuple is associated with equality and/or inequality assertions over variables from the condition column. A tuple is presented only when the associated condition holds. Depending on the variable assignments (valuations) that make the conditions true, a single conditional table maps to many regular tables, also called possible worlds. The complete set of regular relational operators ($\sigma, \bowtie, \pi, \cup, -$) is extended to work on conditional tables naturally, notably, Projection π is the same as relational projection except that the conditional column can never be projected out, Selection $\sigma_{A=a}(T)$ retains all tuples in T and conjugates $t(A) = a$ to the condition of tuple t, and Join $T \bowtie T'$ is obtained by composing tuples from T and T' while jointly considering conditions from T and T', and the join attribute constraints (two tuples agree on valuations of the join attributes).

For example, in Table III: R is a regular table holding a network's routing state that consists of the network's current candidate routes (without loss of generality, we only show a fragment of the content); P_1 is a conditional table representing the policy that determines the network's best route: the variables x, y, z range over possible destination and path values, and the conditions specify two high-level intentions — (1) a static route $[ABC]$ for 1.2.3.4 and (2) a filter that prohibits 1.2.3.5. P_1 represents all the routing states that selects $[ABC]$ for 1.2.3.4 but avoids 1.2.3.5, and the actual state at a particular snapshot is determined by the routes available then. This can be computed by joining R and P_1 ($R \bowtie P_1$).

B. User-Defined Functions and Aggregates

The conventional conditional tables with variables and (in)equality are still too restricted for the rich semantics of network policies, and our plan is to investigate new constructs. To show that this is feasible, we sketch in the following possible extensions with aggregation and user-defined functions, and illustrate how these extensions give a concise representation of one of the most widely used routing policy — the shortest path policy. The forwarding state must follow, among a collection of alternatives paths (to a common destination), the one with shortest distance (hop-counts). This intent translates to the one single tuple in P_2 shown in Table IV: by allowing a user-defined aggregation function s that returns the lowest hop counts among all alternatives paths. The shortest path actually selected — a specific forwarding state compliant with this policy — will be jointly determined by the policy (P_2) and the currently available routes (computed by $R \bowtie \pi_{dest,path,s(path)}(R)$, denoted by R'), which translates to $R' \bowtie P_2$.

P_2	dest	path	s(path)	
	x	y	z	l(y)$\leq$ z

R'	dest	path	s(path)
	1.2.3.4	[ABC]	3
	1.2.3.4	[ADEC]	3
	1.2.3.5	[ABE]	3
	1.2.3.6	[ABE]	3

$\pi_{dest,path}(R' \bowtie P_2)$	dest	path
	1.2.3.4	[ABC]
	1.2.3.5	[ABE]
	1.2.3.6	[ABE]

TABLE IV: P_2: shortest path policy; $R' = R \bowtie \pi_{dest,path,s(path)}(R)$: forwarding state R augmented with smallest hop counts; (bottom) forwarding state selected by P_2.

Next, we illustrate the strength of our knowledge representation system in facilitating policies (intents) that have been poorly supported even with the simple foregoing extensions. Using a load balancer as an example: the intent of a load balancer is to split traffic to two remote servers (1.2.3.4 and 5.6.7.8) over two outgoing paths ([ABC] and [ADC]). Existing approaches to such a load balancer, to the best of our knowledge, are very imperative. One has to articulate a specific "implementation" (e.g., sending traffic to 1.2.3.4 via [ABC] while traffic to the other server via the [ADC] path). This approach also effectively excludes the other possible implementation (i.e. swapping traffic allocation over the two paths). In contrast, we can explicitly specify the high-level intent with knowledge-driven policies — just avoid overloading any of the outgoing paths, immaterial of which server is assigned which path. This is conveniently captured by P_3 in Table V with a functional dependency (FD) $dest \rightarrow \{path, flag\}$: The flag variables u, v and the associated conditions prevent both traffic flows from being assigned to the same path, and the functional dependency ensures a unique path assignment for each server; note that this representation captures both legitimate forwarding states as shown in I_1 and I_2. As a second example, consider a stateful firewall that prohibits external traffic to an internal server (1.2.3.4) unless the server initiated communication first. We can translate this to the dependency between two routes, as shown in P_4, that routes (from some node x) to the server are allowed only when some route from the server (indicating outgoing traffic) to x is presented.

C. Network Addressing

We note that network addresses are not "atomic" entities that can be referenced by constants or variables supported in the foregoing conditional table representation (e.g., P_1 in Table III). Instead, due to scalability concerns (the sheer size of IPv4 is 2^{32}), they — e.g., IP prefix or SDN headers with wildcards — denote sets of "atomic" addresses. Consequently, when the header attribute is referred to in a policy (or a forwarding state), the header values that are different can still overlap, and some form of disambiguation is performed — longest prefix match for IP and priority for SDN. To

P3	dest	path	flag	
	1.2.3.4	[ABC]	u	u = 1
	5.6.7.8	[ABC]	u	u $\neq$ 1
	1.2.3.4	[ADC]	v	v = 1
	5.6.7.8	[ADC]	v	v $\neq$ 1

P4	dest	source	path	
	x	y	w	
	1.2.3.4	x	z	y=1.2.3.4

I1	dest	path
	1.2.3.4	[ABC]
	5.6.7.8	[ADC]

I2	dest	path
	1.2.3.4	[ADC]
	5.6.7.8	[ABC]

TABLE V: `P3`: balances traffic to `1.2.3.4` and `5.6.7.8` on two outgoing paths (`ABC` and `ADC`); `I1` and `I2`: two equally legitimate forwarding states that implement `P3`; `P4`: stateful firewall policy that allows routes to an internal destination `1.2.3.4` only when an outbound route from `1.2.3.4` was initiated first.

support such address attributes in conditional tables, one possibility is to employ some pre-processing procedure that transforms the header variables/constants into disjoint sets, in a straightforward manner: for IP/SDN, simply subtract the more specific/higher priority header from the less specific/lower-priority ones, respectively. This approach leaves the addresses processing out of the policy representation system, and has the potential of independent evolution of addressing and policies. This workaround, however, can miss the opportunity of further optimization with network addresses that a knowledge representation system may offer.

We explore two alternatives that handle addresses within the representation system: one incorporates native support for sets into the representation system, and the other leverages Satisfiability Modulo Theories (henceforth referred to as SMT) solver to handle sets. To allow set in a conditional table and support (set) operation within, we draw upon the insights of extended relational algebra — how the SQL operators originally designed for regular relations were extended to accommodate variables and conditions [18]. In the following, we show the idea by using the relational join ($\bowtie$) as an example. In the extended relational algebra, a join $T \bowtie T'$, where T, T' are conditional tables, is obtained by composing each tuple $t \in T$ with each tuple $t' \in T'$ like in the original SQL join, but with the exception that the new tuple $t \cdot t'$ will have condition $t(\phi) \wedge t'(\phi) \wedge \sigma(t, t')$, where $t(\phi)$ and $t'(\phi)$ are the conditions carried in t and t' respectively, and $\sigma(t, t')$ states that the variables and constants appearing in the join attributes of t, t' must agree on their values. Similarly, we can add set support to $\bowtie$ as follows: when IP prefix appears in the join attributes, whether as a variable or constant, we extend the value agreement requirement to set intersection. An example is shown in Table VI. We believe the same method can be used to extend the other relational operators, namely projection (π) and selection (σ).

An alternative to support set-based network addresses in the conditional tables is to encode the IP address values by the condition column: rather than directly give the address values, whether as constants or a specific set (prefix), we use constrained variables, the condition part of which will be handled by an external constraint solver. For example, for every IP address (i.e. constant c), we replace it by a variable (x) and an associated condition ($x = c$), and then leverage

T1	dest	path
	200.23.16.0/20	[ABC]
	u	[ABD]

T2	dest	type	
	v	customer	
	200.23.20.0/23	w	w$\neq$provider

$T_1 \bowtie T_2$	dest	path	type	
	200.23.16.0/20∩v	[ABC]	customer	
	200.23.20.0/23	[ABC]	w	w$\neq$provider
	u∩v	[ABD]	customer	
	u∩200.23.20.0/23	[ABD]	w	w$\neq$provider

TABLE VI: Extending relational join ($\bowtie$) to support IP prefix by set operation (intersection).

SMT solving that already handles sets. Similarly, for every prefix (set s), replace it by a variable (x) and the associated condition ($x \in s$), as shown in Table VII.

T1	dest	path	
	u	[ABC]	u∈1.2.3.0/24

T2	dest	type	
	v	customer	v∈ 1.2.3.4/28

$T_1 \bowtie T_2$	dest	path	type	
	u	[ABC]	customer	u∈1.2.3.4/28

TABLE VII: Extending join by constraint solving.

With the native set support, we will be able to incorporate custom set operation algorithm and continue to optimize the implementation. Representation of IP addresses with the SMT-based solution, although less direct, allows us to take advantage of the already highly optimized constraint solving with sets. In § VII, we profile and analyze the set-based implementation with the Z3 SMT solver [36]–[38].

D. Extended Relational Algebra

P5	dest	path	
	1.2.3.4	x	x = [ADC]

I3	dest	path
	1.2.3.4	[ABC]
	5.6.7.8	...

P1 ∪ P5	dest	path	
	1.2.3.4	x	x = [ABC]
	1.2.3.4	x	x = [ADC]
	y	z	y $\neq$ 1.2.3.5 $\wedge$ y $\neq$ 1.2.3.4

I4	dest	path
	1.2.3.4	[ADC]
	5.6.7.8	...

TABLE VIII: When `P5` applied in parallel to `P1` , the static route set for `1.2.3.4` can be either `[ABC]` or `[ADC]`, as illustrated in the two instances `I3` and `I4`.

To illustrate the basics of modular composition with knowledge-driven policies, consider policy P_5 in Table VIII, as opposed to P_1 in Table III: P_5 sets up an alternative static route [ADC] for 1.2.3.4. When both P_1 and P_5 are permitted, i.e. when P_1 and P_5 are applicable in *parallel*, their combined effects are represented by a simple union $P_1 \cup P_5$, a new conditional table that prescribes two alternative static routes for 1.2.3.4. Correspondingly, the new composite $P_1 \cup P_5$ permits two instances I_3 and I_4. Observe how our knowledge-driven policies enable the admin to understand policies composed in parallel by simply examining the conditional table obtained by relational $\cup$.

Policies may also be combined *sequentially*, i.e. applied in a strict order, the preceding policy having a higher priority. For example, consider the *sequential* application of the static routes and filtering policy P_1 and the load balancer policy P_3 (Table V). By joining P_1 and P_3 ($P_1 \bowtie P_3$) as shown

P₁⋈P₃	dest	path	flag	
	1.2.3.4	x	u	x=[ABC] ∧ u=1
	~~1.2.3.4~~	~~x~~	~~v~~	~~x=[ABC] ∧ x=[ADC] ∧ v=1~~
	~~y~~	~~z~~	~~u~~	~~y=1.2.3.4 ∧ y≠1.2.3.4 ∧ z=[ABC] ∧ u=1~~
	y	z	u	y=5.6.7.8 ∧ z=[ABC] ∧ u≠1
	~~y~~	~~z~~	~~v~~	~~y=1.2.3.4 ∧ y≠1.2.3.4 ∧ z=[ADC] ∧ v=1~~
	y	z	v	y=5.6.7.8 ∧ z=[ADC] ∧ v≠1

I_5	dest	path
	1.2.3.4	[ABC]
	5.6.7.8	[ADC]

I_6	dest	path
	5.6.7.8	[ABC]

TABLE IX: When P_1 and P_3 are applied sequentially, the join table gives intuitive explanation of the combined policies, as shown by the permitted instances I_5 and I_6.

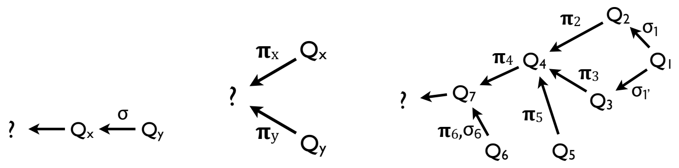

Fig. 2: (left) sequential composition; (middle) parallel composition; (right) example.

in Table IX, the proposed knowledge-driven policies give a natural explanation of sequential composition: $P_1 \bowtie P_3$ produces a conditional table of 6 rows, out of which three have a contradictory condition and are thus struck out. The three remaining rows give a straightforward expression of the combined intention: (1) traffic to 1.2.3.4 can take the explicit path [ABC], but only when that path is not simultaneously taken by 5.6.7.8; (2) traffic to 5.6.7.8 can freely take [ADC]. Correspondingly, the two possible forwarding states are shown in I_5 and I_6. Note that the I_2 state allowed by load balancer policy P_3 is now prohibited by P_1 and is replaced by I_6.

More generally, we propose to use the extended relational operators $\{\pi, \sigma, \bowtie, \cup, -, \rho\}$ as an intuitive composition interface: as shown in Figure 2 (left), the sequential composition of two policies (Q_x, Q_y), producing a composite policy that sequentially applies Q_x and Q_y to a network state, is obtained by relational join ($\bowtie$). Optionally, the higher-priority policy Q_x may use relational selection (σ) to select from Q_y a subset that it accepts. Thus we have $Q_x \bowtie \sigma(Q_y)$; On the other hand, two policies taking effect in parallel in Figure 2 (middle) are combined by union $\cup$: optionally, projection (π) may be used to transform the policy schemas to ensure proper union, thus we have $\pi_x(Q_x) \cup \pi_y(Q_y)$. Put everything together, as illustrated in the example in Figure 2 (right), a composition expressed can be naturally formed with the familiar relational algebra as $(\pi_4((\pi_2(\sigma_1(Q_1) \bowtie Q_2) \cup \pi_3(\sigma_{1'}(Q_1) \bowtie Q_3) \cup \pi_5(Q_5)) \bowtie Q_4) \cup \pi_6(\sigma_6(Q_6))) \bowtie Q_7$.

The SQL-like composition interface has the potential to significantly lower the bar for policy management. In the context of policy routing within an administrative domain using BGP protocol, the nodes in Figure 2 (right) can correspond to the BGP policies configured at individual BGP routers, and our knowledge composition will allow the admin to precisely predict (compute) the policy of the network as a whole, a long thought-after question that remained unanswered until now; In SDNs, on the other hand, knowledge-driven policy composition offers an attractive alternative: rather than forcing the users to adopt and internalize a (more likely a set of) home-grown domain specific languages, we rely on the familiar and generic relational algebra.

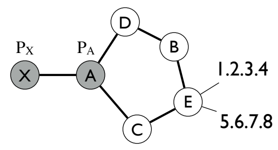

Fig. 3: Policy interaction in interdomain routing: the realizable policies of X are constrained by the policies made at A.

V. Local Policy and Exchange: An Application

In the previous section, we illustrate the possibility of coordinating knowledge-driven policies from a single central point. But in a distributed environment like inter-domain routing, policies are made by independent domains (autonomous systems, or ASes). In fact, the border gateway protocol (BGP) [12], [23], [39]–[43], the de-facto Internet routing protocol, takes the extreme position of favoring autonomy — policies are made based on local preferences within each domains without conforming to any globally-agreed criteria. This extreme approach can lead to unwanted interaction between policies when the choices available to one domain are inadvertently decreased by the decisions made by another. Consider routing policies for 1.2.3.4 and 5.6.7.8 in Figure 3: As X requires routes no more than 3 hops to 1.2.3.4, it is perfectly compatible with its upstream neighbor A whose policy is to balance traffic from its neighbors C and D. That is, a valid path [XACE] that simultaneously satisfies the policies of X and A does exist. However, unaware of the preference of X, A might end up choosing [ADBE] for 1.2.3.4 and the shorter [ACE] for 5.6.7.8, a decision that, though compliant with A's local policy, will render X's policy unsatisfiable.

We do not seek a comprehensive and/or best solution to the foregoing problem — there are potentially many solutions if proper technical and business-level changes can be made to the current BGP system, each with their pros and cons. Instead, we focus on one specific technical issue we termed as policy exchange, and use it to highlight the potential of the proposed knowledge-driven policies. In Figure 3: for A to make a routing selection that is compliant with local policy while "permitting" the neighbor X's policy, A needs to take into account the "impact" of X's policy — the path to 1.2.3.4 should not exceed 2 hops (since A is one hop closer to the destination). Such policy impact, thanks to our proposed knowledge-driven policies, can be explicitly represented, computed, and exchanged to enable informed policy making. As shown in Table X, the original policies of X and A are stored in P_X and P_A; the impact of X on A is to consider paths with no more than 2 hops as represented by $P_{X'}$ ("_" denotes value that does not matter). Once we obtain $P_{X'}$, it can be advertised to A, and merged with A's local policy P_A to form $P_{A'}$ — for the matching records join the conditions ($l(u) \leq 2$ and $x = 1$, $l(u) \leq 2$ and $y = 1$). The new policy $P_{A'}$ jointly

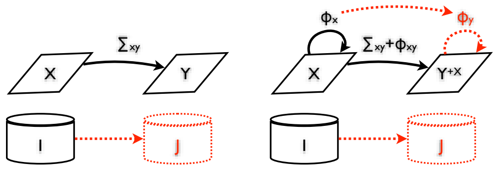

Fig. 4: From schema mapping to policy mapping: (left) extend target schema to include variables referenced by constraints in the source; (right) extend schema mapping to describe relationship between source constraints and target constraints.

reflects A's local concern and takes into account the policy requirement of neighbor X. And the key policy question here is how to compute $P_{X'}$? In other words, how to transform P_X into $P_{X'}$?

P_X	dest	path	
	1.2.3.4	x	l(x)≤3

$P_{X'}$	dest	nh	flag	path	
	1.2.3.4	_	_	u	l(u)≤2

P_A	dest	nh	flag	
	1.2.3.4	C	x	x=1
	1.2.3.4	D	y	y=1
	5.6.7.8	C	x	x≠1
	5.6.7.8	D	y	y≠1

$P_{A'}$	dest	nh	flag	path	
	1.2.3.4	C	x	u	x=1∧l(u)≤2
	1.2.3.4	D	y	u	y=1∧l(u)≤2
	5.6.7.8	C	x	v	x≠1
	5.6.7.8	D	y	v	y≠1

TABLE X: Policy exchange: `P`$_X$ and `P`$_A$ represent the local policies of `X` and `A`; `P`$_{X'}$ represents impact of `X` on `A`. Merging `P`$_{X'}$ into `P`$_A$ gives `P`$_{A'}$, a new policy that forces A to take into account the requirement of `X`.

We argue that the answer is affirmative for knowledge-driven policies, by extending the data exchange paradigm [27]–[29], [34]. Data exchange, one of the oldest data problems, as shown in Figure 4 (left), is concerned with transforming a given source data instance (sets of tables with the source schema X, denoted by I) to a target instance (a new set of tables conforming to the target schema Y, denoted by J), so that the target data "reflects" the source as accurately as possible, while satisfying the source-target mapping Σ_{XY}. Σ_{XY}, also called schema mapping specification, often uses some logical formalism to describe how the content from the source relates to the data content in the target, or vice versa. Policy transformation is concerned with a similar problem: given a policy of X (P_X), finding its impact on Y ($P_{X'}$) is to transform the original policy P_X to adhere to Y's schema, while satisfying the relationship between X and Y. Policy exchange uses the richer conditional tables, whereas conventional data exchange framework only handles regular tables (factual data). In the following, as shown in Figure 4 (right), we sketch our extensions for conditional tables to enable policy transformation:

The key idea is to accommodate the new conditions that are introduced in the conditional tables, denoted by ϕ_X in the source (I) and ϕ_Y in the target (J): first we note that some variables referenced by the source constraint ϕ_X may appear in columns that only appear in the source schema (X) but not in the target (Y). To make sure such constraints are accurately reflected in the target, we extend the target schema Y to include all attributes in X, denoted by Y^{+X}. Using the transformation of P_X (source policy) to $P_{X'}$ (target) in Figure 3 as an example, the schema of A's policy `(dest,next_hop,flag)` is extended to `(dest,next_hop,flag,path)` to include `path` variables. Second, we note that schema mapping Σ_{XY} in conventional data exchange only specifies correspondence between "regular" data, but not those between the constraints in the condition column. Hence, we add a new component to describe how ϕ_X maps to ϕ_Y, denoted by Φ_{XY}. Using specification similar to the standard logical formalism in conventional data exchange, for example, policy P_X can be mapped to $P_{X'}$ by the following logical statements:

```
ΣXX′ : PX(d,x) → (∃ n,f)P′X(d,n,f,x′) ∧ x=Aox′ % for
     any destination d, every path x at node X is
     learned from some path x′ at node A with some next
     hop n and flag f.
ΦXX′ : l(x)≤3 → l(x′)≤2 % maximum hop count of 3 at X
     is reduced to 2 at A because A is one hop closer
     to the destination.
```

VI. Implementation

As a prototype implementation of relational policies, we developed a set of Python functions summarized in Table XI.

	APIs	Lines	Usages
primitives	relational operators ($\sigma, \pi, \bowtie$)	254	query, composition
	IP-prefix handling	95	
	poss(s,p)	208	policy evaluation
applications	poss_s(s,p,ds)	101	BGP simulation
	poss_p(s,p,dp)	64	policy exchange

TABLE XI: Prototype implementation in Python with Z3.

Primitives. We first identify a minimal set of primitives that provides the necessary relational policy functions, and upon which new features can be built. This minimal set includes a relational algebra engine and a policy analyzer (evaluator): the relational algebra engine is the SELECT(σ), PROJECT(π), and JOIN($\bowtie$) operators that take as input policy tables — Python list in main memory, and outputs the resulting policy list by evaluating the conditions via the Python API of Z3 [38] constraint solver. Besides the classic conditional table computation, we hard-coded the aggregates (shortest path evaluation), represent user-defined functions (e.g., hop count) as pre-defined assertions in Z3, and implement IP prefix by 64-bit bit-vector — similar to HSA where two 0/1 bits to encode a single bit in IP prefix with three possible values 0, 1, and * — in both Z3 (when IP prefix appears in a condition) and Python (in table).

The policy analyzer is a `poss(s,p)` predicate that takes as input a network state `s` and a policy `p`, stored in two Python lists, and determines whether there exists a possible world of `p` in which the facts (tuples) of `s` are all true, by systematically exploring valuations of variables in `p` with Z3. We emphasize that we make `poss(s,p)` a primitive because it validates a relational policy (`p`) on a given network state (`s`), a question that must be answered before relational policies can be used in a network. More importantly, the `poss` function (and its incremental variants) provides a building block that is common to many relational policy applications.

Rapid Innovations. As a demonstration of rapid innovation enabled by relational policies, we implemented several appli-

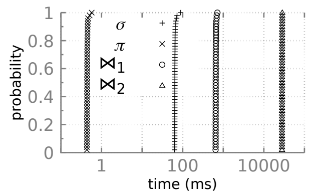

Fig. 5: Relational algebra processing time.

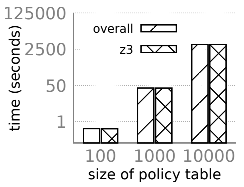

Fig. 6: Scalability property of relational algebra.

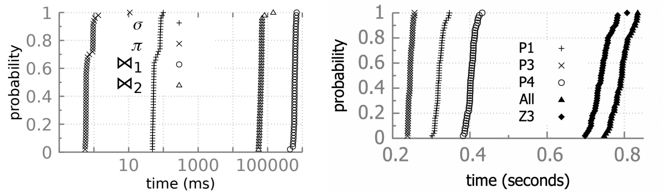

Fig. 7: (left) IP prefix operations; (right) Processing time breakdown of poss(s,p): s contains 80 routing paths; p consists 31 tuples of static routing (P1), 40 tuples of load balancers (P3), and 30 tuples of stateful firewall (P4).

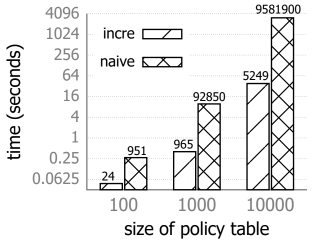

Fig. 8: Scalability property of `poss_s(s,p,ds)`.

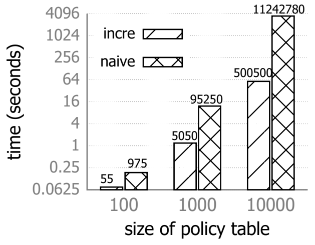

Fig. 9: Scalability property of `poss_p(s,p,dp)`.

cations. *BGP simulation* checks whether an incoming route announcement shall replace the current best route and is implemented by `poss_s(s,p,ds)`. `poss_s` is an incremental variant of `poss(s,p)` that, assuming `p` (current routing policy) permits `s` (current best route), checks whether a delta change `ds` (route announcement) to `s` will still produce a best selection under `p`. Similarly, `poss_p(s,p,dp)`, a second incremental variant of `poss`, implements *Policy exchange* as described in § V: assuming `p` permits `s`, `poss_p(s,p,dp)` checks whether a delta change `dp` to policy `p` will still admit `s`. Both variants only checks the delta against a small set of "relevant" policy/state, exploiting the fact that network policies on disjoint destinations are often independent.

VII. Evaluation

Route View BGP data — RIBs and UPDATEs — are used to generate synthetic policies and realistic network topologies (as observed by a BGP speaker), the two Route View collectors are `route-views2.oregon-ix.net` and `route-views3.oregon-ix.net`, and all files are taken on February 1, 2021 at 00:00 PST. BGP data from the first collector is used in all experiments, the second only in BGP simulation. All experiments were ran on a 64-bit laptop with AMD Ryzen 7 4800H CPU and 15.4G RAM.

A. Primitive Operations

Extended Relational Algebra. We first profile and analyze performance of the relational algebra on symbolic destinations (replacing IP prefix with integers). Figure 5 plots the processing time (repeated 50 times) for each relational operator: σ (selection) and π (projection), using 1000 synthetically generated tuples; $\bowtie_1$ applies shortest path policy P_2 to a path table; $\bowtie_2$ joins two synthetically generated policy tables, both with 1000 tuples, following the pattern of P_1 and P_3. As expected, the π and σ delays are negligible, both $\leq$ 90 ms. The joins incur larger delays, and are slowest when both input tables are conditional tables (policies), but the majority still completes within 28439 ms. In all cases, our Python implementation imposes small delay while Z3 remains the dominating source. On average, Z3 takes 56% of the runtime for $\bowtie_1$ and 96% of the runtime for $\bowtie_2$. Since P_2 has only a single entry, Z3 is only called some number of times on the magnitude of the size of R' for $\bowtie_1$ while $\bowtie_2$ has Z3 called a number of times depending on the magnitude of the size of P_1 and P_3.

We also profile the scalability property of relational algebra, as shown in Figure 6, on the most expensive join ($\bowtie_2$ of two policies tables) on three policy sizes: while the processing time does grow exponentially, our implementation can handle policy of 10,000 entries in $\leq$ 4164 sec. As explained earlier, due to Z3 taking up 96% of the runtime for $\bowtie_2$, the time due to Z3 makes up most of the overall time. Finally, Figure 7 (left) repeats the performance of the relational operators on realistic destinations by embedding IP prefixes taken from the BGP RIB. Compared to Figure 5, the processing delay is significantly larger, because we indexed the integer destinations in the symbolic case, but not with the set based IP prefix, which explodes the Z3 calling time.

Policy Evaluation. Figure 7 (right) plots for `poss` the processing time breakdown on a policy that includes the following: a set of simpler static routing policy (P_1) completes first, the more complex firewall (P_4) takes the longest time, and load balancer sits in between. The overall processing time is within $\leq$ 0.84 second, Z3 being the dominating source of delay. Figure 8 and Figure 9 plot on varying input sizes how the two incremental variants of `poss` significantly reduce processing time: when incrementally evaluating state change (`poss_s`), processing time reduces from 0.268 to 0.047 sec, yielding a reduction rate of 82.5%, thus making the incremental time much lower apart than the naive time (processing time for `poss`). Incrementally evaluating policy change (`poss_p`) shows similar trend but a more drastic reduction, because a single policy change (update, insertion, or deletion) has a higher impact than state change on processing delay. So, here as well, the incremental time is significantly lower than the naive time. In all experiments, the number of Z3 calls are also labeled, giving trend — in terms of scalability property and reduction rate — similar to the processing delay.

The implementation of JOIN takes advantage of hashmaps to keep track of the indices where constants appear in one of the ctables to be joined, so that when we compare tuples in that ctable to tuples in the other table, we can easily exclude tuples where there are mismatch of constants in corresponding

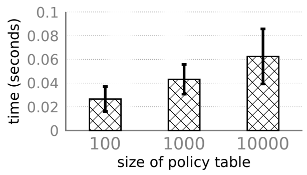

Fig. 10: BGP simulation.

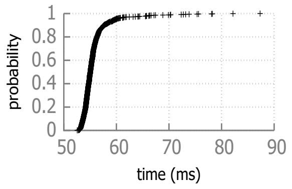

Fig. 11: Policy re-validation after policy exchange.

tuples and gain some performance boost. This same method cannot be done with IP-JOIN with P_1 and P_3. Since with IP-JOIN we do not consider equality for destinations, using a hashmap to store those indices in the same way doesn't make sense. Additionally, in P_1 each entry in the path column is a Z3 variable and not a constant. This means that we cannot determine in the same way which pair of tuples to exclude when matching. So when joining tables of size 1k by 1k, we must make a million comparisons, calling Z3 each time.

B. Applications

BGP Simulation and Policy Exchange. We show the use of poss_s and poss_p in BGP simulation and policy exchange, respectively. First, we use poss_s to simulate BGP operation as described in § VI. Figure 10 plots the average route processing on three policy sizes (100, 1k, and 10k), the error bar shows variance. In all cases, the policy tables are static routing and load balancer policies generated from the RIB file as in primitive operations. And each delta state is a BGP announcement extracted from the collector's UPDATEs collected in the same day. For each policy size, we repeat poss_s on 1500 BGP updates, all complete within .1 second even on the largest policy size.

Next, we plot poss_p overhead during policy exchange — checking a network state after importing a policy fragment generated from BGP RIBs as follows: we randomly select two routeview collectors, denoted by X and A in the following, to be the two nodes exchanging policies in a way similar to Figure 3. Using the RIBs of A, we generate 652 tuples of load balancer policies by randomly pick two paths of different lengths. Next, using the RIBs of X, we generate 163 tuples of path length constraint policies by looking for BGP feeds that simultaneously (1) have an overlapping destination, and (2) have a path length that is between the two counterpart paths stored in the load balancer. Figure 11 shows promising result that more than 95.4% of the policy exchanges can be re-validated in $\leq$ 60 ms.

VIII. Discussion

For relational policies to make an impact, we outline some of the issues that we believe to be rewarding and urgent:

Native Implementation with Relational Database Management System (RDBMS). The performance of our relational policies with main-memory lists and list operations is promising but not without flaws: while our implementation adds little processing delay where the Z3 is the dominating source, the memory usage poses a serious bottleneck as the size of network topology and the number of prefixes in policies grow. Compared to main-memory list, a more robust alternative is to use existing relational database management systems (RDBMS) that are known to easily handle large tables. Previous network state management with databases [44]–[46] all exhibited promising result. In particular, the postgres [47] SDN controller [45] can handle millions of prefixes and the largest Rocketfuel topology with single-digit ms per flow operation. The native use of a highly optimized database also has the added benefit of accelerating operations on relational policies: for example, the incremental evaluation of policies (e.g.,poss_p and poss_s) frequently queries the policy or state to extract the relevant deltas before constraint solving with Z3. As such, our immediate next step is to re-implement relational policies as a native extension to a RDBMS.

Interfacing with Network System. Traditional databases use SQL as a natural data sublanguage in application programs (host language), but the interfacing of relational policies with a network system is not straightforward: policy-carrying attributes (parameters) are often scattered across diverse protocols and/or control software, set and used in many locations and/or steps. For example, BGP features enormous amount of policy knobs that can be tuned to influence route decision. "Install"-ing a relational policy in BGP requires refactoring into the usual BGP attributes — e.g., local preference, MED — to properly enforce route preference or filters [12], [48]–[51]. Our hope is that there may exist a single point (attribute) — such as local preference in BGP — during the whole policy making process that overrides all other aspects, so that synthesizing that particular point alone would be adequate.

A Practical Foundation for Compelling Applications. Whether relational policies can fulfill the potential to accelerate innovations also depends on whether we can keep its overhead under control. While encouraging, the performance of policy tables in this paper is obtained on policies that are largely destination-based (i.e. destination acts as a key). But such assumption does not always hold, policies in Multi-Protocol Label Switching (MPLS) are attached to the topology (end to end paths) — can we still achieve fast processing in the absence of the destination key? On the other hand, the new service of policy exchange relies on user-supplied mappings between policy sources — to what extent can we efficiently automate this process? Just as the relational database replaced the non-relational ones only after its performance catch up, thanks to optimization with indexing and hashing, the success of relational policies depends on identifying compelling policies and their uses that are, at the same time, computationally tractable.

IX. Related Work

We have discussed in § I a range of network modeling and specification techniques and compared them to the proposed relational approach. This section discusses two more areas that help place relational policies in the research community: one prominent effort from the networking community that we consider complementary, and the other stressing the co-evolution of database and networking.

Inter-domain Routing Protocols and Architectures. To enable more flexible policies, many proposals to extend or replace BGP [24]–[26], [52]–[56] were proposed in the past, with the policy components often carefully wired into some clean slate mechanisms, driven by specific needs of a particular party [52], [53], [57]–[61]. But none of these proposals were widely deployed. More recently, D-BGP and Trotsky [62]–[64] examined the simultaneous partial deployments of these protocols, uncovering the architectural features needed to allow their co-existence on the global Internet. Our effort is complementary, we do not seek a total network solution that carefully stitches together policy considerations and routing that works best for all, instead, we focus on the sub-problem of policies. We hope that our solution may become a policy making sub-system which, separated from the connectivity maintenance and route computation sub-system, makes each sub-system easier to manage and independent to evolve, which in turn produces a better policy routing system.

Declarative Networking. In the context of database usages in networking, we are closest to declarative networking [65]–[71] which introduces network datalog — a data query language reminiscent of SQL — as a compact and efficient language for expressing networking such as routing protocols and overlay networks. Network datalog and its variants have proven to be versatile, notable examples being cross tire flow management in SDN control platforms [44], [46], [72]–[74], incremental forwarding state computation in multi-tenant datacenters, and scalable reachability verification for large datacenters. Our work is built on similar belief that database query language marries expressiveness/extensibility and performance/robustness, permits rapid innovation, and has the potential to replace disparate DSLs. But we go beyond programming with definite data, to our best knowledge, we are the first to investigate the use of indefinite data (i.e. conditional tables) to lift policies (intentions) to first order data that can be queried and transformed, to open the door to a new area of policy mediation such as exchange and repair.

X. Conclusion

In this paper, we introduce relational policies, a representation system that lifts network policies as first class data objects, features an intuitive (SQL) user interface, and gives a strong semantics (captures all and only the network states prescribed by the policy). We then present a first of its kind evaluation based on the so-called conditional tables, a complex structure originally invented in the incomplete database research known for its theoretical interest, showing that incremental computation and IP-specific optimization can produce an efficient implementation of relational policies in the narrower context of networking. Moreover, through conditional tables, we illustrate how relational policies can lend itself to novel extensions to a wealth of data mediation techniques that allow us to address long standing problems — e.g., the severely restricted Internet policy routing, and to open the door to new opportunities — e.g., quantitatively repair policies against conflicts. We stress that our focus is not on what policies future networks should support, but rather on arguing that a rigorous and practical policy subsystem can, with some luck, play a more positive role in today's ever-evolving and increasingly more programmable networks.

Acknowledgments. We thank the ICCCN reviewers whose feedback helped improve this paper. This work was supported by National Science Foundation Award CNS-1909450.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[2] X. Jin, J. Gossels, J. Rexford, and D. Walker, "Covisor: A compositional hypervisor for software-defined networks," ser. NSDI'15. USENIX Association, 2015, pp. 87–101.

[3] N. Foster, A. Guha, M. Reitblatt, A. Story, M. J. Freedman, N. P. Katta, C. Monsanto, J. Reich, J. Rexford, C. Schlesinger, D. Walker, and R. Harrison, "Languages for software-defined networks." *IEEE Communications Magazine*, vol. 51, no. 2, pp. 128–134, 2013.

[4] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN Programming with Pyretic," *USENIX ;login*, vol. 38, no. 5, October 2013.

[5] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker, "Composing software-defined networks," ser. nsdi'13. USENIX Association, 2013, pp. 1–14.

[6] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, "Kinetic: Verifiable dynamic network control," ser. NSDI'15. USENIX Association, 2015, pp. 59–72.

[7] 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*.

[8] S. Donovan and N. Feamster, "Intentional network monitoring: Finding the needle without capturing the haystack," ser. HotNets-XIII. New York, NY, USA: Association for Computing Machinery, 2014, p. 1–7.

[9] Y. Yuan, D. Lin, A. Mishra, S. Marwaha, R. Alur, and B. T. Loo, "Quantitative network monitoring with netqre," ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 99–112.

[10] N. Feamster, J. Borkenhagen, and J. Rexford, "Controlling the impact of bgp policy changes on ip traffic," 2001.

[11] T. Wirtgen, Q. De Coninck, R. Bush, L. Vanbever, and O. Bonaventure, "Xbgp: When you can't wait for the ietf and vendors," ser. HotNets '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–7.

[12] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," Internet Requests for Comments, RFC Editor, RFC 4271, 2006.

[13] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," ser. SIGCOMM '16. ACM, 2016, pp. 328–341.

[14] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," RFC 7665, 2015.

[15] P. Zhang, Y. Huang, A. Gember-Jacobson, W. Shi, X. Liu, H. Yang, and Z. Zuo, "Incremental network configuration verification," ser. HotNets '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 81–87.

[16] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 155–168.

[17] R. Birkner, D. Drachsler-Cohen, L. Vanbever, and M. Vechev, "Config2spec: Mining network specifications from network configurations." Santa Clara, CA: USENIX Association, Feb. 2020, pp. 969–984.

[18] S. Abiteboul, R. Hull, and V. Vianu, Eds., *Foundations of Databases: The Logical Level*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[19] S. Abiteboul, P. Kanellakis, and G. Grahne, "On the representation and querying of sets of possible worlds," ser. SIGMOD '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 34–48.

[20] T. Imieliundefinedski and W. Lipski, "Incomplete information in relational databases," *J. ACM*, vol. 31, no. 4, p. 761–791, Sep. 1984.

[21] N. Feamster, H. Balakrishnan, and J. Rexford, "Some foundational problems in interdomain routing," in *In HotNets, 2004. (Cited on*, 2004, pp. 41–46.

[22] T. G. Griffin and G. Wilfong, "A Safe Path Vector Protocol," in *INFOCOM*, 2000.

[23] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE Trans. on Networking*, vol. 10, pp. 232–243, 2002.

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

[25] R. Mahajan, D. Wetherall, and T. Anderson, "Negotiation-based routing between neighboring isps," in *NSDI*, 2005.

[26] ——, "Mutually controlled routing with independent ISPs," in *NSDI*, 2007.

[27] P. Barceló, "Logical foundations of relational data exchange," *SIGMOD Rec.*, vol. 38, no. 1, p. 49–58, Jun. 2009.

[28] P. G. Kolaitis, "Schema mappings, data exchange, and metadata management," ser. PODS '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 61–75.

[29] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "The tsimmis project: Integration of heterogenous information sources," in *Information Processing Society of Japan (IPSJ 1994)*, 1994.

[30] F. N. Afrati and P. G. Kolaitis, "Repair checking in inconsistent databases: Algorithms and complexity," ser. ICDT '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 31–41.

[31] J. Chomicki, "Consistent query answering: Five easy pieces," ser. ICDT'07. Berlin, Heidelberg: Springer-Verlag, 2007, p. 1–17.

[32] M. Arenas, L. Bertossi, and J. Chomicki, "Consistent query answers in inconsistent databases," ser. PODS '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 68–79.

[33] L. Bertossi, "Consistent query answering in databases," *SIGMOD Rec.*, vol. 35, no. 2, p. 68–76, Jun. 2006.

[34] L. E. Bertossi, J. Chomicki, P. Godfrey, P. G. Kolaitis, A. Thomo, and C. Zuzarte, "Exchange, integration, and consistency of data: report on the arise/nisr workshop," *SIGMOD Record (SIGMOD)*, vol. 34, no. 3, pp. 87–90, 2005.

[35] A. Y. Levy, "Logic-based techniques in data integration," 1999.

[36] Yices, "http://yices.csl.sri.com/."

[37] Examples of Yices and Maude Encoding of Routing Policies, http://netdb.cis.upenn.edu/fsr/.

[38] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," ser. TACAS'08/ETAPS'08. Springer-Verlag, 2008, pp. 337–340.

[39] T. Bates, E. Chen, and R. Chandra, "BGP route reflection: An alternative to full mesh internal BGP (IBGP)," RFC 4456, 2006.

[40] P. Traina, D. McPherson, and J. Scudder, "Autonomous system confederations for BGP," RFC 5065, 2007.

[41] T. G. Griffin and G. Wilfong, "An analysis of BGP convergence properties," in *SIGCOMM*, 1999.

[42] L. Gao, T. G. Griffin, and J. Rexford, "Inherently safe backup routing with BGP," in *IEEE INFOCOM*, Apr. 2001.

[43] L. Gao and J. Rexford, "Stable Internet routing without global coordination," in *ACM SIGMETRICS*, 2000.

[44] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang, "Network virtualization in multi-tenant datacenters," ser. NSDI'14. USENIX Association, 2014, pp. 203–216.

[45] A. Wang, X. Mei, J. Croft, M. Caesar, and B. Godfrey, "Ravel: A database-defined network," ser. SOSR '16. ACM, 2016, pp. 5:1–5:7.

[46] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Fml: Practical declarative network management," ser. WREN '09. ACM, 2009, pp. 1–10.

[47] PostgreSQL: The World's Most Advanced Open Source Relational Database, "https://www.postgresql.org/."

[48] T. Griffin and G. T. Wilfong, "Analysis of the MED oscillation problem in BGP," in *ICNP'02*, 2002.

[49] D. R. McPherson and V. Gill, "BGP MULTI_EXIT_DISC (MED) Considerations," RFC 4451, Mar. 2006.

[50] T. Li, R. Chandra, and P. S. Traina, "BGP Communities Attribute," RFC 1997, Aug. 1996.

[51] J. Borkenhagen, R. Bush, R. Bonica, and S. Bayraktar, "Policy Behavior for Well-Known BGP Communities," RFC 8642, Aug. 2019.

[52] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet routing," in *ACM SIGCOMM*, 2009.

[53] X. Yang, D. Clark, and A. W. Berger, "Nira: a new inter-domain routing architecture," *IEEE/ACM Trans. Netw.*, vol. 15, no. 4, 2007.

[54] Y. Wang, I. Avramopoulos, and J. Rexford, "Design for configurability: Rethinking interdomain routing policies from the ground up," *IEEE J.Sel. A. Commun.*, vol. 27, no. 3, pp. 336–348, Apr. 2009.

[55] T. G. Griffin, A. D. Jaggard, and V. Ramachandran, "Design principles of policy languages for path vector protocols," ser. SIGCOMM '03. ACM, 2003, pp. 61–72.

[56] L. Subramanian, M. Caesar, C. T. Ee, M. Handley, M. Mao, S. Shenker, and I. Stoica, "Hlp: A next generation inter-domain routing protocol," ser. SIGCOMM '05. ACM, 2005, pp. 13–24.

[57] D. Zhu, M. Gritter, and D. R. Cheriton, "Feedback based routing," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, p. 71–76, Jan. 2003.

[58] I. Ganichev, B. Dai, P. B. Godfrey, and S. Shenker, "Yamr: Yet another multipath routing protocol," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 5, p. 13–19, Oct. 2010.

[59] H. T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi, "Bananas: An evolutionary framework for explicit and multipath routing in the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 4, p. 277–288, Aug. 2003.

[60] S. Agarwal, Chen-Nee Chuah, and R. H. Katz, "Opca: robust interdomain policy routing and traffic control," in *2003 IEEE Conference onOpen Architectures and Network Programming.*, 2003, pp. 55–64.

[61] K. Argyraki and D. R. Cheriton, "Loose source routing as a mechanism for traffic policies," ser. FDNA '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 57–64.

[62] R. R. Sambasivan, D. Tran-Lam, A. Akella, and P. Steenkiste, "Bootstrapping evolvability for inter-domain routing with d-bgp," ser. SIGCOMM '17. ACM, 2017.

[63] ——, "Bootstrapping evolvability for inter-domain routing," ser. HotNets-XIV. ACM, 2015, pp. 12:1–12:7.

[64] J. McCauley, Y. Harchol, A. Panda, B. Raghavan, and S. Shenker, "Enabling a permanent revolution in internet architecture," ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–14.

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

[66] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica, "Declarative networking: Language, execution and optimization," ser. SIGMOD '06. ACM, 2006, pp. 97–108.

[67] T. Condie, J. M. Hellerstein, P. Maniatis, S. Rhea, and T. Roscoe, "Finally, a use for componentized transport protocols," in *In HotNets IV*, 2005.

[68] Y. Mao, B. T. Loo, Z. Ives, and J. M. Smith, "Mosaic: Unified declarative platform for dynamic overlay composition," ser. CoNEXT '08. ACM, 2008, pp. 5:1–5:12.

[69] X. Chen, Z. M. Mao, and J. van der Merwe, "Towards automated network management: Network operations using dynamic views," ser. INM '07. ACM, 2007, pp. 242–247.

[70] C. Liu, L. Ren, B. T. Loo, Y. Mao, and P. Basu, "Cologne: A declarative distributed constraint optimization platform," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 752–763, Apr. 2012.

[71] C. Liu, B. T. Loo, and Y. Mao, "Declarative automated cloud resource orchestration," ser. SOCC '11. ACM, 2011, pp. 26:1–26:8.

[72] 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," ser. OSDI'10, 2010.

[73] 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.*, vol. 47, no. 1, pp. 15–26, Jan. 2017.

[74] M. Casado, N. Foster, and A. Guha, "Abstractions for software-defined networks," *Commun. ACM*, vol. 57, no. 10, pp. 86–95, Sep. 2014.