Fauré: A Partial Approach to Network Analysis

Fangping Lan, Bin Gui
Temple University

Anduo Wang*
Temple University

ABSTRACT

Formal analysis has been intensively studied (e.g., deep customization and synergistic co-design) in the networking domain, but one assumption remains largely unexamined: there is a *complete* evaluation that expects definite knowledge of the task, and is expected to output a decisive result. This paper argues for a "partial" approach, a departure from the de facto, to network analysis in a practical environment with uncertain events and limited visibility. Specifically, we seek (1) loss-less modeling in which network uncertainty is explicitly handled without corrupting the querying capability; and (2) complete verification relative to the level of information available, which reaches an inconclusive result only when more information is needed. As a realization of this vision, we present *fauré*, a preliminary design in which a datalog extension (called *fauré-log*) for incomplete information is developed to enable loss-less modeling, and combined with static analysis of pure datalog to implement example relative-complete verifiers.

ACM Reference Format:
Fangping Lan, Bin Gui and Anduo Wang. 2021. Fauré: A Partial Approach to Network Analysis. In *The Twentieth ACM Workshop on Hot Topics in Networks (HotNets '21), November 10–12, 2021, Virtual Event, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3484266.3487391

1 INTRODUCTION

Formal analysis has made significant progress in the networking domain in the past decade. A spectrum of methods (simulation [18, 42], model checking [48], SMT solving [19], etc) were adapted, benefiting networks ranging from enterprise networks and SDNs [32–34] to datacenters and private WANs [30, 55, 59], and addressing tasks from basic reachability [58] to more advanced features (middleboxes, network updates [55, 61], protocol interactions [1] etc). Underlying all these successes is a common workflow — often taken for granted — in which a *comprehensive* evaluation is performed on an *entirely known* network. The critical assumption is that the task is definite and the analysis result is decisive. While this complete approach is salient in classical formal analysis, it can be ill-fitted for networking.

First, formal analysis was originally designed for a single definite input (e.g., hardware verification), but networking often involves exponential number of states that can possibly occur. A moderate sized control plane can result in a huge number of data planes that easily exceed the capability of highly optimized data plane verifiers [32, 33]. Failures — an important aspect that must be factored into the verifier as many important errors occur only under specific failures [20] — can quickly blow up the search engine even on small topology. Despite the many efforts — e.g., avoiding generating individual data planes under failures [15, 20], abstracting networks jointly controlled by multiple protocols [6, 20, 46] — to treat the symptoms, existing models rarely include explicit constructs for the causes — uncertainty events such as failures or network interactions that cause the problem. Leaving out uncertainty is understandable due to limited support in formal analysis, nevertheless, it can be a missed opportunity for networking.

A second mismatch between the complete approach and networking is that our visibility into a network is often limited. In the global Internet, the inability to obtain the BGP configuration inputs from external domains leaves most attempts to verify the global routing behavior futile [5, 16, 17, 21–23, 45, 50, 54, 57]. While some argue that the public transit has been in decline [4, 26, 28], the domain structure organized around domains is unlikely to change any time soon [44, 52, 53], meaning that visibility into the inter-domain will remain constrained. Even within a single domain, when management is shared by multiple participants [12, 49], a complete network view is unlikely due to performance concerns. In all these cases, even when some aspects of the network are unknown, it is desirable to implement some (perhaps weaker) verification than stop working entirely.

Informed with these observations, we take a departure from the de-facto complete analysis and propose a *partial* approach for the networking environment. We seek to (1) achieve accurate network modeling even with uncertainty

*Lead author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HotNets '21, November 10–12, 2021, Virtual Event, United Kingdom
© 2021 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-9087-3/21/11…$15.00
https://doi.org/10.1145/3484266.3487391

(e.g., arbitrary failures), which we call *loss-less modeling*; and to (2) perform complete verification on networks that are only partly known, which we call *relative-complete verification.* More precisely, loss-less modeling seeks a representation for uncertain networks such that it assures sound and complete reasoning like in the case of a definite network. In relative-complete verification, the goal is to develop new tests that fully use the available information such that if the tests reach an indecisive conclusion, a more powerful verification requiring more information is absolutely needed.

To illustrate the feasibility of our approach, we develop a preliminary design called *fauré*, inspired by a representation system in incomplete database called conditional table (or c-table) [2, 3, 29, 51, 56]. C-tables use variables to represent unknown values whose meaning are given by a condition of boolean atoms: a single c-table models many possible networks (i.e. an uncertain network); more importantly, it is loss-less in the sense that the difference between the c-table and the possible instances it represents is not visible to (an extended) relational algebra — any SQL query over a c-table will produce exactly the same answer as on the corresponding (possible) regular tables. Our first contribution is to turn the useful algebraic tool of c-tables into a deductive one that is more suitable for analysis. To this end, we develop a datalog extension — a deductive counterpart of extended SQL — to correctly reference and valuate the variables in a c-table, forming our own partial specification language called *fauré-log*.

Fauré-log supports loss-less modeling out-of-box: As we show in reachability analysis under link failures, the *fauré-log* gives a general (datalog-like) query interface to access and manipulate the c-table representation of a partial network state under a link failure pattern without introducing visible corruption, that is, *fauré-log* query on a single partial network is guaranteed to be equivalent to iteratively querying all possible networks. By combining the new *fauré-log* valuation with static analysis readily available in pure datalog, *fauré* also lends itself to relative-complete verification: As we show in the management of a network by multiple teams, *fauré-log* enables us to develop two complete tests relative to two levels of information visible to the verifier. In the first level where only (the definition of the) constraints are known, we formulate a test as constraint subsumption, a datalog problem of program containment that can be further automated by a novel reduction into query evaluation in *fauré-log*; In the second level where the update is also known, the update information is incorporated into the test by a systematic constraint rewrite in *fauré-log*. A practical implementation of *fauré-log* in PostgreSQL and encouraging evaluation results on realistic forwarding configuration are also presented.

2 FAURÉ OVERVIEW

Fauré is a partial network analysis toolkit with two loosely-coupled components: loss-less modeling and relative-complete verification. Loss-less modeling embodies a precise condition that we believe should be satisfied in any meaningful model of uncertain events or unknown information, that is, information corruption introduced by the new partial expressions should not be visible in reasonably constructed queries. We achieve this by adopting c-tables — the relational structure of incomplete database — as the data model, and developing a general query language called *fauré-log*. *Fauré-log* extends the classic datalog evaluation strategy to reference partial networks represented in the c-tables. Using data plane verification under link failures as an example, we illustrate how *fauré-log* offers loss-less modeling out-of-box.

In the relative-complete verification component, instead of a single conclusive verifier, we implement two tests: the weaker test will succeed whenever a decisive answer is permitted by the least amount of information, and return with "I don't know" only when more information is needed. When the additional information becomes known, the second (stronger) test capable of processing it can be invoked. As we show in policy-compliance checking during updates in an enterprise network managed by multiple teams, we leverage query containment analysis readily available in datalog to check policy when we only know the policy definition itself; and exploit query rewrite to further incorporate network updates when they become available. Notably, the new *fauré-log* valuation strategy for c-tables reduces query containment in pure-datalog to query evaluation in *fauré-log*.

	fauré-log	modeling	verification
c-tables	✓	✓	✓
constraint			✓
query evaluation	✓	✓	✓
update rewrite			✓
query containment			✓

Table 1: *Fauré*'s enabling techniques.

Table 1 summarizes the new techniques: *fauré-log* incorporates the c-tables by a new evaluation strategy (§ 3) and enables loss-less modeling (§ 4). On top of *fauré-log*, constraint as 0-ary queries over c-tables, constraint rewrite, and query containment are combined to achieve relative-complete verification (§ 5).

3 FROM DATALOG TO FAURÉ-LOG

The data model of *fauré* is inspired by the c-table [2, 3, 29, 56], a relational structure for incomplete information that is particularly attractive because it does not introduce corruption visible to any SQL query — through a straightforward extension to the relational algebra. The algebraic extension, however, does not give the clean semantics and valuation strategies critical to network analysis. To this end, we develop our own datalog extension call *fauré-log*.

C-table and Why SQL/pure-datalog Fall Short

A c-table allows variables to occur in the table entries and introduces an additional column of conditions over the variables. The idea is to use the variables to denote information with proper attributes but currently unknown, and to use the conditions to characterize legitimate information. An example is $\mathtt{P}^\mathtt{i}$ in Table 2, where the first tuple says the destination `1.2.3.4` has a path that is either `[ABC]` or `[ADEC]`, the second tuple says a destination other than `1.2.3.4` uses path `[ABE]`, and the third says `1.2.3.6` uses `[ADEC]` no matter what (empty condition). Depending on the instantiation of the variables, $\mathtt{P}^\mathtt{i}$ corresponds to many different regular network states, one possible instance is `P`.

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

$\mathtt{P}^\mathtt{i}$	dest	path	
	1.2.3.4	$\bar{x}$	$\bar{x}$ = [ABC]∨ $\bar{x}$ = [ADEC]
	$\bar{y}$	[ABE]	$\bar{y}$≠1.2.3.4
	1.2.3.6	[ADEC]	

C	path	cost
	[ABC]	3
	[ADEC]	4
	[ABE]	3

Table 2: PATH is a regular database with regular tables `P`, `C` (PATH={`P`, `C`}); PATH' is a *fauré* database that includes a c-table $\mathtt{P}^\mathtt{i}$ (PATH'={$\mathtt{P}^\mathtt{i}$, `C`})

The c-tables can be queried by a straightforward extension of SQL. For example, the join of two c-tables $\mathtt{T}_1$ and $\mathtt{T}_2$ can be obtained by concatenating every tuple $\mathtt{t}_1 \in \mathtt{T}_1$ and $\mathtt{t}_2 \in \mathtt{T}_2$ and associating it with $\varphi_1 \wedge \varphi_2 \wedge \varphi(\mathtt{t}_1, \mathtt{t}_2)$ where φ_1 (φ_2) is the condition associated with $\mathtt{t}_1$ ($\mathtt{t}_2$), and $\varphi(\mathtt{t}_1, \mathtt{t}_2)$ is a condition that states the equality between join attributes in $\mathtt{t}_1$ and $\mathtt{t}_2$. While convenient for ad-hoc data retrieval, the algebraic approach of SQL is inadequate for program analysis (e.g., query containment in relative-complete verification 5). Thus we argue for the deductive approach of datalog.

It turns out that datalog extension for c-tables is not readily available and far less obvious. A specific difficulty is how to evaluate a datalog-like program over a c-table. How to map variables in the datalog program to the underlying domain of the c-tables which have variables in themselves? To see why this is difficult and to prepare technically for the development of *fauré-log*, we first review datalog and its valuation following notions in [2, 9]. A datalog query q over a (usual) database schema $\mathbf{R}$ is a finite collection of rules of the form

$$\mathtt{H}(\mathtt{u}) : -\mathtt{B}_1(\mathtt{u}_1), \cdots, \mathtt{B}_n(\mathtt{u}_n). \tag{1}$$

where $\mathtt{B}_\mathtt{i}$'s are relation (predicate) names in $\mathbf{R}$, and `H` is a relation not in $\mathbf{R}$; and `u` and $\mathtt{u}_\mathtt{i}$'s are free tuples that can use either constants in the attribute domain of $\mathbf{R}$ (denoted by **dom**) or variables (we use *var(q)* to denote all variables appeared in q). The subexpression $\mathtt{B}_1(\mathtt{u}_1), \cdots, \mathtt{B}_n(\mathtt{u}_n)$ is the *body* of the rule, and $\mathtt{H}(\mathtt{u})$ is the *head*. A rule generates new facts by variable valuation — a function (denoted by v) from *var(q)* to **dom**: if one can find values that hold for the body, then one can derive the head. Thus, the meaning of a query can be defined by variable valuations. Let q be a datalog query given by the foregoing rules, and let $\mathbf{I}$ be a database instance of $\mathbf{R}$, the image (query result) of $\mathbf{I}$ under q is

$$q(\mathbf{I}) = \{v(\mathtt{u}) | v \text{ is a valuation and } v(\mathtt{u}_\mathtt{i}) \in \mathbf{I}\} \tag{2}$$

```
q1: ANS(z) :- P('1.2.3.4',y), C(y,z). /* what is the path
    cost for destination 1.2.3.4? */
q2: ANS(z)[φ∧ x='1.2.3.4'] :- P^i(x,y)[φ], C(y,z),
    x='1.2.3.4'. /* equivalent query using explicit
    comparison (=) */
q3: ANS(z)[φ] :- P^i('1.2.3.5',y)[φ], C(y,z). /* implicit
    pattern matching over c-table */
```

Listing 1: First attempt: datalog query over c-tables

For example, running query $\mathtt{q}_1$ (in Listing 1) on database **PATH** (in Table 2) gives $\mathtt{q}_1(\mathbf{PATH}) = \{\langle 3 \rangle\}$. Observe that a constant occurred in a free tuple produces implicit pattern matching on the underlying domain (**dom**), which is equivalent to explicit comparison in SQL (e.g.,`P('1.2.3.4', y)` is equivalent to `SELECT FROM P WHERE P.dest =' 1.2.3.4'`). Such straightforward pattern matching no longer works once we enter the domain of the c-tables.

Fauré-log and Evaluation over C-tables

We develop *fauré-log* by taking extra care in mediating between variables in the (*fauré-log*) program and variables in the referenced c-tables. These two types of variables play critically different roles: variables in *fauré-log* are referenced by usual $\mathtt{x}, \mathtt{y}, \mathtt{z}, \cdots$ and are simply called variables, variables representing unknown values in the c-tables are denoted by $\bar{\mathtt{x}}, \bar{\mathtt{y}}, \cdots$ and are called c-variables. To handle the new c-variables, the underlying domain of c-tables is extended by incorporating the c-variables into the usual attribute domain (of constants): the intuition is to treat the c-variables as the usual constant values, only that they are currently "unknown". We call this new domain the c-domain, denoted by $\mathbf{dom}^\mathbf{C}$. Similar to pure datalog, we define the *fauré-log* query q^F over c-tables with schema $\mathbf{R}^\mathbf{F}$ as a finite collection of rules of the form

$$\mathtt{H}(\mathtt{u})[(\wedge_{\mathtt{i}=1}^{\mathtt{n}} \varphi_\mathtt{i}) \wedge (\wedge_{\mathtt{i}=1}^{\mathtt{m}} \mathtt{C}_\mathtt{i})] : - \\ \mathtt{B}_1(\mathtt{u}_1)[\varphi_1], \cdots, \mathtt{B}_n(\mathtt{u}_n)[\varphi_n], \mathtt{C}_1, \cdots, \mathtt{C}_m. \tag{3}$$

here $\mathtt{B}_\mathtt{i}$'s are relations in $\mathbf{R}^\mathbf{F}$; the $\mathtt{u}, \mathtt{u}_\mathtt{i}$'s are "free" tuples that contain both regular variables and symbols (constants or c-variables) in $\mathbf{dom}^\mathbf{C}$ (e.g.,$\mathtt{P}^\mathtt{i}(\mathtt{x}, \mathtt{y})$, $\mathtt{P}^\mathtt{i}(\mathtt{x}, \mathtt{[ABC]})$, and $\mathtt{P}^\mathtt{i}(1.2.3.4, \bar{\mathtt{x}})$ are all valid expressions); $\varphi_\mathtt{i}$'s are conditions over $\mathbf{dom}^\mathbf{C}$; and $\mathtt{C}_\mathtt{i}$'s are explicit comparisons (i.e.=, ! =, <, > etc) over $\mathbf{dom}^\mathbf{C}$ (e.g.,$\bar{\mathtt{x}}$ = [ABC],x! = 1.2.3.4). Observe that, while hidden in the SQL extensions for c-tables [2, 3, 29], our syntax (equation 3) explicitly reflects the manipulation of the conditions.

A *fauré-log* rule also generates new tuples from tuples matching those in its body. For example, q_2 generates $\{\langle 3[\bar{x} = [ABC]]\rangle, \langle 4[\bar{x} = [ADEC]]\rangle\}$ (i.e. depending on the assignment to $\bar{x}$, the answer is 3 or 4 respectively). In this derivation, x / y maps to 1.2.3.4 / $\bar{x}$ in the c-domain. Note that q_2 and q_1 are equivalent except that q_2 uses explicit equality over variables while q_1 relies on implicit pattern matching. It is desirable to also support such implicit pattern matching in *fauré-log*: for example, in q_3, $P^i(1.2.3.5, y)$ should match the second tuple in P^i, allowing q_3 to derive the answer $q_3(\mathbf{PATH}') = \{\langle 3\rangle\}$.

Unlike the trivial pattern matching in regular tables (a constant always matches itself, in fact v only needs to be defined for the datalog variables), in *fauré-log*, a constant matches itself as well as a c-variable as long as it does not cause contradiction in the condition, and a variable can match any constants and c-variables in the c-domain. Formally, we extend the v to the c-domain by the following rules: (1) *fauré-log* variables are assigned to constants or c-variables (by simple substitution like in v); (2) a constant c is assigned to either itself, or to a c-variable $\bar{x}$ if c does not contradict $\bar{x}$'s condition (e.g.,$P(1.2.3.5, y)$ maps to $P^i(\bar{y}, [ABE])[\bar{y} \neq 1.2.3.4 \wedge \bar{y} = 1.2.3.5]$). We call this the c-valuation function, denoted by v^C. Using v^C, query evaluation on c-tables take exactly the same form as equation 2. Finally, without diving into details, we point out that recursion can be incorporated into *fauré-log* using the usual fixed point approach [2, 9], and the "not" modifier can be added to mean non-derivable from the c-table as in [7]. Both negation and recursion are particularly useful for network analysis as we will see in § 4 and § 5.

4 LOSS-LESS MODELING

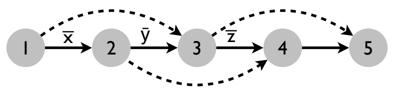

Figure 1: Fast rerouting under link failures: protected links encoded by $(\bar{x}, \bar{y}, \bar{z})$

F	node	node	
	1	2	$\bar{x} = 1$
	1	3	$\bar{x} = 0$
	2	3	$\bar{y} = 1$
	2	4	$\bar{y} = 0$
		...	

R	source	dest	
	1	2	$\bar{x} = 1$
			...
	1	5	$\bar{x} = 1 \wedge \bar{y} = 1 \wedge \bar{z} = 1$
	1	5	$\bar{x} = 0 \wedge \bar{z} = 1$
	1	5	$\bar{x} = 0 \wedge \bar{z} = 0$
	1	5	$\bar{x} = 1 \wedge \bar{y} = 0$
	2	3	$\bar{y} = 1$

Table 3: F represents for all (possible) forwarding behaviors; R represents reachability under combinations of failures.

This section presents loss-less modeling by reachability analysis under link failures. Consider an excerpt of a fast rerouting configuration inspired by [31] (Figure 1): The nodes 1, 2, 3, 4, 5 are abstract addressable routing/forwarding entities, the bold arrows between the nodes show the primary links that we want to protect, and the dashed arrows are backup links that will be used as a detour when failure occurs. The many possible forwarding behaviors, due to arbitrary failures, can be described in a single c-table once and for all. As shown in the c-table F (Table 3): the schema F(node, node) says packets arrived at the first node are to be forwarded to the second; the three c-variables $\bar{x}, \bar{y}, \bar{z} \in \{0, 1\}$ denote the state of the four protected links — 0 means the link fails while 1 means normal.

```
/* reachability as recursive query */
q₄: R(f,n₁,n₂)[φ] :- F(f,n₁,n₂)[φ].
q₅: R(f,n₁,n₂)[φ_F ∧ φ_R] :- F(f,n₁,n₃)[φ_F], R(f,n₃,n₂)[φ_R].
/* examples of failure patterns */
q₆: T₁(f,n₁,n₂)[φ ∧ x̄ + ȳ + z̄=1 ] :- R(f,n₁,n₂)[φ], x̄ + ȳ + z̄
     =1. % reachability under 2-link failure
q₇: T₂(f,2,5)[φ ∧ ȳ = 0] :- T₁(f,2,5)[φ], ȳ=0. %
     reachability between 2 and 5 under 2-link failure, one
     of the failure must be (2,3)
q₈: T₃(f,1,n₂)[φ ∧ ȳ + z̄<2] :- R(f,1,n₂)[φ], ȳ + z̄<2. %
     reachability to 1 with at least 1-link failure
```

Listing 2: *Fauré-log* offers intuitive and flexible reachability analysis under failures for free

To perform reachability analysis under failures, we only need to construct a *fauré-log* query that specifies the failure pattern. For example, all pair-wise reachability analysis can be computed by rules q_4, q_5 in Listing 2, which outputs an R table (a fragment of which is) shown in Table 3. As shown by more examples in Listing 2, *fauré-log* gives a flexible query language for *link failure patterns* as conditions over the c-variables (rules q_6, q_8), and nested queries (rules q_6, q_7 that query the output of another query). Note that R, F in Table 3 are loss-less with respect to *fauré-log*: information that can be queried (via *fauré-log*) from R, F are not distinguishable from enumerating all the concrete data planes.

5 RELATIVE-COMPLETE VERIFICATION

We use a running example — the task of checking whether a network update (or change) can affect the validity of one or more constraints — to illustrate relative-complete verification, because (1) it is a pressing problem as networks (e.g., global-scale private WANs [55, 59]) undergoing frequent and increasingly complicated updates have to continually maintain network-wide constraint, and (2) it allows us to (conveniently) vary the levels of information available to the verifier. In the rest of this section, we incrementally develop two verifiers (verification tests) that assure a constraint continues to hold after a network update, using increasingly more (available) information: (i) the least information is the constraint (definition) alone; and (ii) a more powerful test is possible when both the constraint and the update are known.

We first describe our driving example inspired by [49], the problem of managing an enterprise network by multiple teams in which the constraints are network policies maintained by individual teams or a network-wide invariant, the update is a configuration change, and the test is performed by a (dedicated) separate team. Specifically, the enterprise network connects two frontend subnets and two backend servers, the two subnets are for market management (denoted by `Mkt`) and research & development (`R&D`), and the two backend servers are the critical accounting server (`CS`) and the general purpose server (`GS`). The network is managed by two teams, a security team that maintains firewall configuration and a second team responsible for traffic engineering (TE) that configures the load balancers. The target constraints (T_1, T_2) we want to verify after a network update are: T_1 requires `Mkt` traffic to the critical server `CS` to go through a firewall; T_2 requires `R&D` traffic to all servers to pass through a load balancer.

We can model this network by three c-tables (**Net** = {`R`, `Lb`, `Fw`}): `R(subnet, server, port)` keeps reachability information about traffic (on specific ports) allowed from the subnets to the servers; `Lb(subnet, server)` (resp.,`Fw(subnet, server)`) indicates a load balancer (resp., firewall) is deployed between the subnet and server. The c-domain for the attributes `subnet, server, port` are {`Mkt.`, `R&D`, $\bar{x}$}, {`CS`, `GS`, $\bar{y}$} and {`80`, `344`, `7000`, $\bar{p}$}, respectively ($\bar{x}, \bar{y}$ and $\bar{p}$ are c-variables). Over these tables, we can formulate the constraints T_1, T_2 as *fauré-log* programs (rules q_9, q_{10}) in Listing 3. The general idea is that a (network) constraint can be viewed as a *fauré-log* query whose result is a 0-ary predicate `panic`. If the query evaluates to $\emptyset$, then the constraint holds. Otherwise, the query will produce {`panic`} signaling constraint violation.

Category (i) test: using only network constraints

Category (i) test considers the least information when we are allowed to look only at the constraints themselves. Our only opportunity to use available information is through subsumption of one constraint by one or more other constraints (that are known to hold).

Suppose we know that the TE and security teams each maintain some policies (denoted by constraint C_{lb} and C_s, respectively): C_{lb} requires the TE team to load balance traffic to the critical server — only frontend subnets can send packets to the critical `CS`, and the packets must use port `7000` and pass through a load balancer; C_s requires the security team to control packets to all the servers — the packets must use one of the three ports `80`, `334` and `7000`, and must pass through a firewall. Similar to the encoding of T_1, T_2, we can formulate C_{lb}, C_s as `panic` queries shown in Listing 3. For example, the C_{lb} query derives `panic` from the three possible violations (rules q_{13}-q_{15}). If C_{lb}, C_s are known to hold after the update, to prove that T_1, T_2 will also hold, we only need to show that $\{C_{lb}, C_s\}$ subsume T_1, T_2. Here, $\{C_{lb}, C_s\}$ does subsume T_1 because the corresponding rule q_9 is really just a special case of q_{17}, i.e.$q_9 \subseteq q_{17}$, the violation of T_1 implies the violation of C_s (q_{17}).

But how do we automate constraint subsumption? Now that constraints are 0-ary query programs, constraint subsumption becomes a special case of program containment [2, 25]. Program containment on pure datalog is known to be NP-complete. While the exponential complexity of containment checking [2, 3, 24, 25] may not block practical use since the constraint programs are usually short, *fauré-log* allows us to side step the containment analysis entirely. The idea is a novel reduction of program containment in datalog to query evaluation in *fauré-log*.

```
q9: panic :- R(Mkt,CS,p̄), ¬Fw(Mkt,CS) % T1
q10: panic :- R(R&D,ȳ,7000), ¬Lb(R&D,ȳ) % T2
/* Clb(q13-q15) enumerates 3 TE violations */
q11: panic :- Vt(x,y,p)
q13: Vt(x̄,CS,p̄) :- R(x̄,CS,p̄), x̄ ≠Mkt, x̄ ≠R&D
q14: Vt(x̄,CS,p̄) :- R(x̄,CS,p̄), ¬Lb(x̄,CS)
q15: Vt(x̄,CS,p̄) :- R(x̄,CS,p̄), p̄ ≠7000
/* Cs(q17-q18) enumerates 2 security violations */
q16: panic :- Vs(x,y,p)
q17: Vs(x̄,ȳ,p̄) :- R(x̄,ȳ,p̄),¬Fw(x̄,ȳ)
q18: Vs(x̄,ȳ,p̄) :- R(x̄,ȳ,p̄),p̄ ≠ 80,p̄ ≠344,p̄ ≠7000
```

Listing 3: Constraints as 0-ary *fauré-log* query

We outline the reduction by continuing with our running example: Suppose q_9 produces `panic`, then we must have `R(Mkt, CS,` $\bar{p}$`)` and `Fw(`$\bar{x}, \bar{y}$`)`[$\bar{x}$ = `Mkt` $\wedge$ $\bar{y}$ = `CS`] in the network state. Evaluating q_{17} on such instance (q_{17}(**Net**) as defined in § 3) will produce `panic`. Such evaluation is always possible in *fauré-log* because we can rewrite a constraint rule into a form whose body tuples (like q_9) contain only c-variables or constants ($\bar{p}$ and `CS` in q_9) but not variables (e.g.,`x`). We only need to substitute the variables with c-variables augmented with proper conditions. After such rewrite that removes variables, a variable-free *fauré-log* query `P` contains another query `Q`, if evaluating `P` on the database obtained from the body of `Q` produces `panic`.

Category (ii) test: using both constraints and updates

The containment based category (i) test is only relative-complete, it may reach an inconclusive answer, but more information will enable a more thorough analysis. For example, $\{C_{lb}, C_s\}$ does not subsume T_2, thus category (i) test will return “unknown” on T_2. But if we are allowed to use also the update information, we will be able to perform a test that completes verification for T_2.

To see how this is possible, consider verifying constraint `C` under an update `U`, we can incorporate `U` into `C` by rewriting it into a new constraint `C′`: `C′` holds before the update `U` if and only if `C` holds after the update. That is, we can verify `C` after `U` by checking `C′` that reflects `U`. Continuing with T_2 and Listing 3, suppose we also know that the update (of the

TE team) was to remove load balancing between Mkt and CS, and add load balancing for R&D and GS. To incorporate this update into T_2, we construct a new T_2' by following the rewrite in [37]:

```
/* add (R&D,GS) to the load balancer */
q19: Lb(R&D,GS).
q20: Lb1(x̄,ȳ) :- Lb(x̄,ȳ)
/* delete (Mkt,CS) from the load balancer */
q21: Lb2(x̄,ȳ) :- Lb1(x̄,ȳ)[x̄ ≠Mkt]
q22: Lb2(x̄,ȳ) :- Lb1(x̄,ȳ)[ȳ ≠CS]
/* panic after updating load balancer(Lb2) */
q24: panic :- R(R&D,ȳ,7000), ¬ Lb2(R&D,ȳ)
```

Listing 4: Rewriting constraints to reflect update

where rules $q_{19} - q_{20}$ (resp. $q_{21} - q_{22}$) add (resp. delete) the new (old) tuple to (from) the old Lb configuration. The new relation Lb_2 that reflects the update then substitutes Lb in T_2 (q_{10}) and creates the new constraint T_2' (q_{24}). To ensure T_2 continues to hold, we only need to check $T_2' \subseteq \{C_{lb}, C_s\}$. It turns out that this can be verified by the *fauré-log* query evaluation method as in category (i) test.

6 PRELIMINARY RESULTS

Practical implementation. We implement *fauré-log* in the PostgreSQL database [47]. This is especially important as it allows us to leverage existing database structure (e.g., indexing) to accelerate *fauré-log* evaluation. Our main contribution is to support c-tables by rewriting SQL's default valuation in three steps: (1) generate the data part of a c-table (key terms reserved for c-variables) in pure SQL; (2) add proper conditions (including *fauré-log* pattern matching) by a sequence of SQL UPDATE; and (3) invoke Z3 [14] to remove tuples with contradictory conditions. Note that while Postgres supports native recursion, recursive *fauré-log* is implemented by stratification [2, 25] to correctly process the conditions.

Preliminary evaluation. We evaluate the running time of *fauré-log* queries in listing 2 on realistic forwarding configuration inferred from BGP RIB (route-views2.oregon-ix.net on 2021-06-10). We choose listing 2 because it covers representative features of *fauré-log* such as recursive and nested query. For a given set of prefixes, we generate for each prefix the forwarding entries as follows: randomly select 5 AS paths where one of them is used as a primary link while the rest serve as the backups; set the preference of the backup links (in a random order) so that a backup will be used only when the primary and all the backups with higher preferences have failed. We then perform all pair-wise reachability analysis (result in R) by q_4, q_5, as well as reachability under three failure patterns ($q_6 - q_8$). All experiments are run on a 64-bit laptop with 1.4 GHz CPU and 8 GB memory. Table 4 summarizes the results: on four inputs (# of prefixes from 1000 to 922067), for each analysis except the recursion $q_4 - q_5$ that compute all pair-wise reachability, we show the SQL and Z3 completion time (averaged over 10 runs) separately. The number of tuples generated indicating the size of the analysis is also illustrated. Overall, the SQL running time is encouraging, even on 922067 prefixes (all the prefixes in the RIB file), all pair-wise analysis through recursion complete in < 70 minutes.

	$q_4 - q_5$	q_6			q_7			q_8		
#prefix	sql	sql	Z3	#tuples	sql	Z3	#tuples	sql	Z3	#tuples
1000	0.625	0.85	796.35	42425	0.08	0.27s	16	0.15	12.64	828
10000	5.75	8.96	-	418224	0.27	3.41	194	1.8	137.05	8706
100000	54.85	113.48	-	4435862	1.66	25.22	1387	34.67	1941.04	86360
922067	816.4	4169.02	-	46503247	11.1	288.17	16490	267.05	-	858180

Table 4: Running time (seconds) of reachability analysis on four rib inputs: '-' means over 2 hours.

7 RELATED WORK

Network datalog. Datalog-like language was first introduced in declarative networking [10, 11, 38–41, 43], and later used for network management [8, 13, 27, 35, 36] and verification [18, 42, 60]. To our best knowledge, *fauré-log* is the first to support incomplete information (i.e. c-tables).

Partial representation. Prior work often uses specialized data structure [20] tailored to specific verification task [6] or network topology [46] to locate a (drastically smaller) subset of the network state that is relevant. In contrast, the c-tables at the heart of *fauré* can be accessed and transformed by arbitrary *fauré-log* queries.

Incremental computation. Notable examples include Jinjing [55] that exploits practical heuristics, and INCV [60] that leverages generic engine (e.g., differential datalog), both have the entire network state incrementally maintained. In contrast, *fauré*'s relative-complete verifiers use constraint subsumption, a reasoning process that entirely eliminates the need to access network state.

8 CONCLUSION

This paper argues for a *partial* approach to network analysis when our knowledge of the network is uncertain or unavailable, a significant departure from the de-facto complete scheme. Central to partial analysis is the notion of *loss-less modeling* that accurately models network uncertainty, and *relative-complete verification* that reaches an inconclusive result only when more information is absolutely needed. As a realization of this vision, we present *fauré*, a preliminary design in which, a datalog extension called *fauré-log* is developed to access and manipulate partial network states, and various static analysis is combined with *fauré-log* evaluation to reason about incomplete network information. Practical implementation of *fauré* and encouraging evaluation are also presented.

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

REFERENCES

[1] Anubhavnidhi Abhashkumar, Aaron Gember-Jacobson, and Aditya Akella. 2020. Tiramisu: Fast Multilayer Network Verification. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 201–219. https://www.usenix.org/conference/nsdi20/presentation/abhashkumar

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

[3] Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. 1987. On the Representation and Querying of Sets of Possible Worlds. In *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data (SIGMOD âĂŹ87)*. Association for Computing Machinery, New York, NY, USA, 34âĂŞ48. https://doi.org/10.1145/38713.38724

[4] Hari Balakrishnan, Sujata Banerjee, Israel Cidon, David Culler, Deborah Estrin, Ethan Katz-Bassett, Arvind Krishnamurthy, Murphy McCauley, Nick McKeown, Aurojit Panda, Sylvia Ratnasamy, Jennifer Rexford, Michael Schapira, Scott Shenker, Ion Stoica, David Tennenhouse, Amin Vahdat, and Ellen Zegura. 2021. Revitalizing the Public Internet by Making It Extensible. *SIGCOMM Comput. Commun. Rev.* 51, 2 (May 2021), 18âĂŞ24. https://doi.org/10.1145/3464994.3464998

[5] Anindya Basu, Chih-Hao Luke Ong, April Rasala, F. Bruce Shepherd, and Gordon Wilfong. 2002. Route oscillations in I-BGP with route reflection. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications (SIGCOMM '02)*. ACM, New York, NY, USA, 235–247. https://doi.org/10.1145/633025.633048

[6] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2018. Control Plane Compression. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 476âĂŞ489. https://doi.org/10.1145/3230543.3230583

[7] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. 2011. Answer Set Programming at a Glance. *Commun. ACM* 54, 12 (Dec. 2011), 92âĂŞ103. https://doi.org/10.1145/2043174.2043195

[8] Martin Casado, Nate Foster, and Arjun Guha. 2014. Abstractions for Software-defined Networks. *Commun. ACM* 57, 10 (Sept. 2014), 86–95. https://doi.org/10.1145/2661061.2661063

[9] S. Ceri, G. Gottlob, and L. Tanca. 1989. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering* 1, 1 (1989), 146–166. https://doi.org/10.1109/69.43410

[10] Xu Chen, Z. Morley Mao, and Jacobus van der Merwe. 2007. Towards Automated Network Management: Network Operations Using Dynamic Views. In *Proceedings of the 2007 SIGCOMM Workshop on Internet Network Management (INM '07)*. ACM, New York, NY, USA, 242–247. https://doi.org/10.1145/1321753.1321757

[11] Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Sean Rhea, and Timothy Roscoe. 2005. Finally, a use for componentized transport protocols. In *In HotNets IV*.

[12] Kevin Dackow, Andrew Wagner, Tim Nelson, Shriram Krishnamurthi, and Theophilus A. Benson. 2020. Solver-Aided Multi-Party Configuration. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 103âĂŞ109. https://doi.org/10.1145/3422604.3425944

[13] Bruce Davie, Teemu Koponen, Justin Pettit, Ben Pfaff, Martin Casado, Natasha Gude, Amar Padmanabhan, Tim Petty, Kenneth Duda, and Anupam Chanda. 2017. A Database Approach to SDN Control Plane Design. *SIGCOMM Comput. Commun. Rev.* 47, 1 (Jan. 2017), 15–26. https://doi.org/10.1145/3041027.3041030

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

[15] Seyed K. Fayaz, Tushar Sharma, Ari Fogel, Ratul Mahajan, Todd Millstein, Vyas Sekar, and George Varghese. 2016. Efficient Network Reachability Analysis Using a Succinct Control Plane Representation. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, Savannah, GA, 217–232. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/fayaz

[16] Nick Feamster, Ramesh Johari, and Hari Balakrishnan. 2007. Implications of Autonomy for the Expressiveness of Policy Routing. *IEEE/ACM Trans. Netw.* 15, 6 (Dec. 2007), 1266–1279. https://doi.org/10.1109/TNET.2007.896531

[17] Nick Feamster and Jennifer Rexford. 2007. Network-Wide Prediction of BGP Routes. *IEEE/ACM Trans. Netw.* 15, 2 (April 2007), 253âĂŞ266. https://doi.org/10.1109/TNET.2007.892876

[18] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A General Approach to Network Configuration Analysis. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, USA, 469âĂŞ483.

[19] Ari Fogel, Stanley Fung, Luis Pedrosa, Meg Walraed-Sullivan, Ramesh Govindan, Ratul Mahajan, and Todd Millstein. 2015. A General Approach to Network Configuration Analysis. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*. USENIX Association, USA, 469âĂŞ483.

[20] Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. 2016. Fast Control Plane Analysis Using an Abstract Representation. In *Proceedings of the 2016 ACM SIGCOMM Conference (SIGCOMM '16)*. Association for Computing Machinery, New York, NY, USA, 300âĂŞ313. https://doi.org/10.1145/2934872.2934876

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

[22] Timothy G. Griffin and Gordon Wilfong. 1999. An Analysis of BGP Convergence Properties. In *SIGCOMM*.

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

[24] Ashish Gupta, Yehoshua Sagiv, Jeffrey D. Ullman, and Jennifer Widom. 1994. Constraint Checking with Partial Information. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '94)*. Association for Computing Machinery, New York, NY, USA, 45âĂŞ55. https://doi.org/10.1145/182591.182597

[25] Alon Y. Halevy, Inderpal Singh Mumick, Yehoshua Sagiv, and Oded Shmueli. 2001. Static Analysis in Datalog Extensions. *J. ACM* 48, 5 (Sept. 2001), 971âĂŞ1012. https://doi.org/10.1145/502102.502104

[26] Yotam Harchol, Dirk Bergemann, Nick Feamster, Eric Friedman, Arvind Krishnamurthy, Aurojit Panda, Sylvia Ratnasamy, Michael Schapira, and Scott Shenker. 2020. A Public Option for the Core. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 377âĂŞ389. https://doi.org/10.1145/3387514.3405875

[27] Timothy L. Hinrichs, Natasha S. Gude, Martin Casado, John C. Mitchell, and Scott Shenker. 2009. FML: Practical Declarative Network Management. In *Proceedings of the 1st ACM Workshop on Research on Enterprise Networking (WREN '09)*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/1592681.1592683

[28] Geoff Huston. [n. d.]. The Death of Transit and Beyond. https://labs.apnic.net/presentations/store/2017-02-28-death-of-transit.pdf. ([n. d.]).

[29] Tomasz Imieliński and Witold Lipski. 1984. Incomplete Information in Relational Databases. *J. ACM* 31, 4 (Sept. 1984), 761âĂŞ791. https://doi.org/10.1145/1634.1886

[30] Karthick Jayaraman, Nikolaj Bjørner, Jitu Padhye, Amar Agrawal, Ashish Bhargava, Paul-Andre C Bissonnette, Shane Foster, Andrew Helwer, Mark Kasten, Ivan Lee, Anup Namdhari, Haseeb Niaz, Aniruddha Parkhi, Hanukumar Pinnamraju, Adrian Power, Neha Milind Raje, and Parag Sharma. 2019. Validating Datacenters at Scale. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 200âĂŞ213. https://doi.org/10.1145/3341302.3342094

[31] Juniper networks, Fast Reroute Overview. (14-Sep-18). [n. d.]. https://www.juniper.net/documentation/en_US/junos-space-apps/connectivity-services-director4.1/topics/concept/fast-reroute-understanding.html. ([n. d.]).

[32] Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. 2013. Real Time Network Policy Checking Using Header Space Analysis. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation (nsdi'13)*. USENIX Association, USA, 99âĂŞ112.

[33] Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header space analysis: static checking for networks. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12)*. USENIX Association, Berkeley, CA, USA.

[34] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. 2012. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HotSDN '12)*. Association for Computing Machinery, New York, NY, USA, 49âĂŞ54. https://doi.org/10.1145/2342441.2342452

[35] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Natasha Gude, Paul Ingram, Ethan Jackson, Andrew Lambeth, Romain Lenglet, Shih-Hao Li, Amar Padmanabhan, Justin Pettit, Ben Pfaff, Rajiv Ramanathan, Scott Shenker, Alan Shieh, Jeremy Stribling, Pankaj Thakkar, Dan Wendlandt, Alexander Yip, and Ronghua Zhang. 2014. Network Virtualization in Multi-tenant Datacenters. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, Berkeley, CA, USA, 203–216. http://dl.acm.org/citation.cfm?id=2616448.2616468

[36] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. 2010. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation (OSDI'10)*.

[37] Alon Y. Levy and Yehoshua Sagiv. 1993. Queries Independent of Updates. In *Proceedings of the 19th International Conference on Very Large Data Bases (VLDB '93)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 171–181. http://dl.acm.org/citation.cfm?id=645919.672674

[38] Changbin Liu, Boon Thau Loo, and Yun Mao. 2011. Declarative Automated Cloud Resource Orchestration. In *Proceedings of the 2Nd ACM Symposium on Cloud Computing (SOCC '11)*. ACM, New York, NY, USA, Article 26, 8 pages. https://doi.org/10.1145/2038916.2038942

[39] Changbin Liu, Lu Ren, Boon Thau Loo, Yun Mao, and Prithwish Basu. 2012. Cologne: A Declarative Distributed Constraint Optimization Platform. *Proc. VLDB Endow.* 5, 8 (April 2012), 752–763. https://doi.org/10.14778/2212351.2212357

[40] Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. 2006. Declarative Networking: Language, Execution and Optimization. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data (SIGMOD '06)*. ACM, New York, NY, USA, 97–108. https://doi.org/10.1145/1142473.1142485

[41] Boon Thau Loo, Joseph M. Hellerstein, Ion Stoica, and Raghu Ramakrishnan. 2005. Declarative Routing: Extensible Routing with Declarative Queries *(SIGCOMM '05)*. ACM, 12. https://doi.org/10.1145/1080091.1080126

[42] Nuno P. Lopes, Nikolaj Bjørner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. 2015. Checking Beliefs in Dynamic Networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 499–512. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/lopes

[43] Yun Mao, Boon Thau Loo, Zachary Ives, and Jonathan M. Smith. 2008. MOSAIC: Unified Declarative Platform for Dynamic Overlay Composition. In *Proceedings of the 2008 ACM CoNEXT Conference (CoNEXT '08)*. ACM, New York, NY, USA, Article 5, 12 pages. https://doi.org/10.1145/1544012.1544017

[44] James McCauley, Yotam Harchol, Aurojit Panda, Barath Raghavan, and Scott Shenker. 2019. Enabling a Permanent Revolution in Internet Architecture. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM âĂŹ19)*. Association for Computing Machinery, New York, NY, USA, 1âĂŞ14. https://doi.org/10.1145/3341302.3342075

[45] D. McPherson, V. Gill, D. Walton, and A. Retana. RFC 3345, 2002. Border Gateway Protocol (BGP) Persistent Route Oscillation Condition. (RFC 3345, 2002).

[46] Gordon D. Plotkin, Nikolaj Bjørner, Nuno P. Lopes, Andrey Rybalchenko, and George Varghese. 2016. Scaling Network Verification Using Symmetry and Surgery. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '16)*. Association for Computing Machinery, New York, NY, USA, 69âĂŞ83. https://doi.org/10.1145/2837614.2837657

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

[48] Santhosh Prabhu, Kuan Yen Chou, Ali Kheradmand, Brighten Godfrey, and Matthew Caesar. 2020. Plankton: Scalable network configuration verification through model checking. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 953–967. https://www.usenix.org/conference/nsdi20/presentation/prabhu

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

[50] B. Quoitin and S. Uhlig. 2005. Modeling the routing of an autonomous system with C-BGP. *IEEE Network* 19, 6 (2005), 12–19. https://doi.org/10.1109/MNET.2005.1541716

[51] Removed for anonymous submission. [n. d.]. ([n. d.]).

[52] Raja R. Sambasivan, David Tran-Lam, Aditya Akella, and Peter Steenkiste. 2015. Bootstrapping Evolvability for Inter-Domain Routing. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets-XIV)*. ACM, New York, NY, USA, Article 12, 7 pages. https://doi.org/10.1145/2834050.2834101

[53] Raja R. Sambasivan, David Tran-Lam, Aditya Akella, and Peter Steenkiste. 2017. Bootstrapping Evolvability for Inter-domain Routing with D-BGP *(SIGCOMM '17)*. ACM, New York, NY, USA, 14. https://doi.org/10.1145/3098822.3098857

[54] Renata Teixeira and Jennifer Rexford. 2004. A Measurement Framework for Pin-Pointing Routing Changes. In *Proceedings of the ACM SIGCOMM Workshop on Network Troubleshooting: Research, Theory and Operations Practice Meet Malfunctioning Reality (NetT '04)*. Association for Computing Machinery, New York, NY, USA, 313âĂŞ318. https://doi.org/10.1145/1016687.1016704

[55] Bingchuan Tian, Xinyi Zhang, Ennan Zhai, Hongqiang Harry Liu, Qiaobo Ye, Chunsheng Wang, Xin Wu, Zhiming Ji, Yihong Sang, Ming Zhang, Da Yu, Chen Tian, Haitao Zheng, and Ben Y. Zhao. 2019. Safely and Automatically Updating In-Network ACL Configurations with Intent Language. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM '19)*. Association for Computing Machinery, New York, NY, USA, 214âĂŞ226. https://doi.org/10.1145/3341302.3342088

[56] Ron van der Meyden. 1998. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems (the book grow out of the Dagstuhl Seminar 9529: Role of Logics in Information Systems, 1995)*, Jan Chomicki and Gunter Saake (Eds.). Kluwer, 307–356.

[57] Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. 2000. Persistent route oscillations in inter-domain routing. *Computer Networks* 32, 1 (2000), 1 – 16. https://doi.org/10.1016/S1389-1286(99)00108-5

[58] G.G. Xie, Jibin Zhan, D.A. Maltz, Hui Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. 2005. On static reachability analysis of IP networks. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, Vol. 3. 2170–2183 vol. 3. https://doi.org/10.1109/INFCOM.2005.1498492

[59] Fangdan Ye, Da Yu, Ennan Zhai, Hongqiang Harry Liu, Bingchuan Tian, Qiaobo Ye, Chunsheng Wang, Xin Wu, Tianchen Guo, Cheng Jin, Duncheng She, Qing Ma, Biao Cheng, Hui Xu, Ming Zhang, Zhiliang Wang, and Rodrigo Fonseca. 2020. Accuracy, Scalability, Coverage: A Practical Configuration Verifier on a Global WAN. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 599âĂŞ614. https://doi.org/10.1145/3387514.3406217

[60] Peng Zhang, Yuhao Huang, Aaron Gember-Jacobson, Wenbo Shi, Xu Liu, Hongkun Yang, and Zhiqiang Zuo. 2020. Incremental Network Configuration Verification. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 81âĂŞ87. https://doi.org/10.1145/3422604.3425936

[61] Peng Zhang, Xu Liu, Hongkun Yang, Ning Kang, Zhengchang Gu, and Hao Li. 2020. APKeep: Realtime Verification for Real Networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 241–255. https://www.usenix.org/conference/nsdi20/presentation/zhang-peng