Indirect Network Troubleshooting with The Chase

Paper #47, 6 pages

ABSTRACT

The future of static verification in networking may be obscured by two clouds: the complexity of distributed systems with highly concurrent events, and the decision-making on infrastructures growing without a premeditated plan. This paper discusses a possible solution to these issues, in which the huge space of analyzing distributed systems and the macro-questions of system evolution are addressed by a common structure, a logical implication problem which we call *indirect troubleshooting*. The usefulness and feasibility of indirect troubleshooting is illustrated by a preliminary realization with *the chase*, a remarkable process for mechanically deciding implications.

…The beauty and clearness of the dynamical theory, which asserts heat and light to be modes of motion, is at present obscured by two clouds.

The Lord Kelvin, Nineteenth Century Clouds over the Dynamical Theory of Heat and Light

1 INTRODUCTION

Network diagnosis and troubleshooting tools [5, 11, 12, 19, 21, 22, 25, 29, 43] used to be the most advanced aid in debugging networks just as testing once was in software, until the rise of software-defined networking (SDN), when static verification [1, 7–9, 17, 18, 20, 24, 26–28, 38, 41, 42, 46, 48, 50–52] took off. What static verification lacks in the breadth at first — e.g., early tools unanimously focusing on reachability — it makes up in completeness. By comprehensively exercising the network on an abstract model (e.g., symbolically testing all forwarding paths in a proper data structure), static verification establishes the "existence" of a target invariant in the network, even when the violation at runtime is latent or rare. As network models become more expressive (admitting richer semantics), the exploration algorithms faster (exploiting symmetry etc.), and new services (e.g., coverage metrics [8, 47]) emerge, static verification seems to be on a parallel route to dominate network analysis as program verification subsumed testing. This bright future, however, may be obscured by two clouds.

Cloud I. Networks are by nature distributed systems with highly concurrent events. This key complexity is to some extent defined away by SDN, and program verification[1] alike. Take the popular verification primitive (a typical invariant) "does the program (network snapshot) as exercised by this test input (data packet) exhibit that characteristic?" as an example: when adopted in networking, the primitive translates to "instantaneous verification" of the network at a particular time instance while at a definite state (e.g., forwarding), a task that treats distribution (e.g., route announcements from a neighbor) and concurrency (e.g., link failure) as external environments, which are often abstracted away in the network model and left out during model exploration. While a single run of the primitive completes the verification task on a centralized sequential software, fully checking every relevant network "incarnation" may involve (potentially exponentially) many runs. Not surprisingly, several recent developments [20, 50] have started treating such pain points.

Cloud II. Networks are an ever-growing live infrastructure. Unlike circuits, chips, or software development, where static verification is applied to a clean-slate with well-defined goals in a pre-fixed life cycle, network infrastructures continue to operate and evolve, under constant failures, over a span of decades, without a well-laid plan. If static verification tools were to facilitate decision making for such infrastructures, the network operators, oftentimes non-programmers, would have to retrofit macro-questions like "Is it safe (or impossible) to add a new invariant (policy) given existing ones? Is this failure a sure symptom of this concern? Is an error detected here also signaling a potential failure in the remote end of the network?" into lower-level verification primitives for detailed packets, paths etc. While similar inference problem based on some notion of causality was addressed in several early troubleshooting systems [11, 12, 19], a more rigorous formulation with static verification, however, is still missing.

In this paper, we propose a solution to the issues with the two clouds by combining existing techniques such as instrumentation/monitoring [19, 21, 43], history based and statistical inference [5, 25, 29, 49], and traditional static verification with a new primitive we call *indirect troubleshooting*. Indirect troubleshooting decides if known facts about the network $\sum$ imply some other unknown property γ, denoted $\sum \vdash \gamma$. $\sum = \{\sigma_1, \sigma_2, \cdots\}$ is a set of "premises" we begin with, obtained from existing tools that check the network states *directly*. These premises may describe an easily verifiable policy about a network partition, a shared belief about a subset of failures, or a detected error signaling a larger fault, to name a few examples. γ is the "conclusion" we seek to *indirectly* derive. It expresses a more comprehensive property about the network as a whole that existing tools cannot assess. This can include an evaluation of policy changes to aid network evolution, an assertion of the impact of faults to guide network diagnosis, and more.

While the general decision problem $\sum \vdash \gamma$ in mathematical logic is not decidable, there is a solution to deciding the implication among a useful class of invariants known as data dependencies (or simply dependencies) in databases. This ranges from SQL key dependencies to arbitrary logical

[1]A rich literature on verifying distributed systems does exist. Here we focus on program verification designed for sequential programs, which is the main inspiration for network verification in the SDN era.

sentences over tables. Almost all useful dependencies can be represented in a simple structure called a *tableau* (plural: *tableaux*), where the dependency is described by an example, such as a template scenario. The tableau representation can be used to decide implications using a process called *the chase* [39]. For indirect troubleshooting, the conclusion γ can be mechanically modified to reflect another dependency (premise), σ, using the chase process i.e. $\texttt{chase}(\gamma, \sigma)$, which can be used to check if σ implies γ.

Inspired by the chase, we illustrate how the ambitious goal of indirect troubleshooting can be feasibly realized. Specifically, we tackle two decision problems: the *temporal decision problem* and the *spatial decision problem.* In the temporal decision problem, we aim to verify if a property that holds in one network environment also holds in other environments (e.g., failure scenarios). In the spatial decision problem, we seek to verify a network-wide invariant (e.g., reachability) based on known premises (e.g., verified forwarding policies at each node), possibly under separate partitions of the network. We examine the issues with *Cloud I* as decision problems by concrete examples in § 2. We then attempt to solve the decision problems by using *fauré* [32], which can consider network dynamics in its reasoning process. However, we show that *fauré* fails to reuse existing premises and cannot be used with disparate network views, encouraging a paradigm shift to indirect troubleshooting (§ 3). In § 4, we demonstrate that the temporal decision problem can be solved using *the chase.* To address the spatial decision problem, we develop a method that applies the chase process on tableaux with disparate views. Furthermore, we establish that the spatial decision problem subsumes the temporal decision problem. Next, we speculate how the chase may enable a flexible framework to cover issues of *Cloud II* (§ 5), and quantify the benefits of the chase by preliminary results (§ 6).

2 MOTIVATING EXAMPLES

This section uses two concrete examples to detail the issues of *Cloud I*; as we will see in § 4, they also drive our preliminary design of indirect troubleshooting with the chase.

Concurrent events

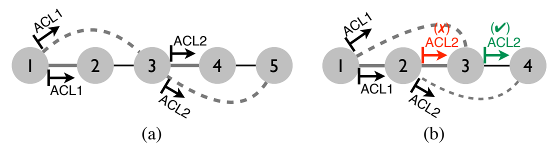


Figure 1: Correct configuration for each link failure (consistent access control on each pair of primary link and its backup) does not ensure consistency in the network as a whole.

To see the impact of highly concurrent events in troubleshooting networks, consider Figure 1 (a): when a backup link is added to protect a "primary" link, it is common to add a backup link with equivalent access control, for example the primary link $1-2$ ($3-4$) in solid line and its backup $1-3$ ($3-5$) in dashed line are assigned the same access control `ACL1` (`ACL2`); ensuring multi-path consistency for each link failure. Unfortunately, this correct configuration per-event (link failure) does not always guarantee consistency on the network as a whole. Consider the slightly modified topology in Figure 1 (b): though each backup link (1-3 and 2-4) has an `ACL` that matches its primary ($1-2$ and $2-3$, respectively), inconsistency still arises. The path 1-3-4 (when $1-2$ fails) is configured with `ACL1` but 1-2-3-4 is associated with both `ACL`s. The correct configuration should instead associate `ACL2` with $3-4$, a link that is neither a primary nor a backup.

When it comes to troubleshooting, checking multi-path consistency for a single link failure event is relatively straightforward. For example, if we're dealing with the protected (and primary) link $1-2$, we only need to consider two scenarios (i.e., link $1-2$ up or down) in the affected area, only focusing on nodes 1, 2, and 3, while ignoring the rest of the network. However, the task becomes increasingly complex when we need to verify consistency for all events, as we must consider the entire network, and the number of scenarios grows exponentially with the number of concurrent events (e.g., link failures). Therefore, it's helpful to determine whether verifying individual events is sufficient to verify their combinations, which can be done via indirect troubleshooting (i.e. the temporal decision problem).

Distributed invariants

Another source of complexity is the distributed management of network policies (invariants). Consider Figure 2 (a) with three invariants: one security policy that blocks packets from `10.2` to `10.4`, implemented by filters at 1 and 2; and two rewriting policies that change the source header at 1 and destination header at 2 as shown in the figure. Notably, 1 and 2 can be viewed as two separate management partitions, in each of which the packet filter is properly enforced alongside a legitimate rewrite. But these two *correct* partitions, when combined, does not securely block traffic. Figure 2 (b) shows the trace of a packet originated at `10.2` with source `10.2` and destination `10.3` that successfully bypasses the filters and eventually reaches `10.4`, revealing a hard-to-predict interaction between the distributed invariants.

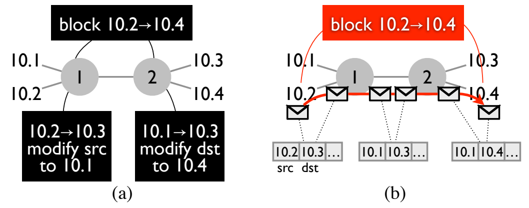


Figure 2: Correct configuration for each network partition (filtering properly enforced for node 1 and 2 separately) does not ensure intended packet filtering on the global network.

From a management and troubleshooting perspective, distributing the enforcement of invariants can help scale the management of complex networks. By managing and reasoning about a smaller partition, which consists of a subset of network elements with a subset of concerns, the task becomes simpler compared to managing the entire network, every invariant, and their interactions simultaneously. However, this practice also introduces a new challenge: As a collection of invariants correctly implemented in each partition separately, it becomes essential to determine what properties still hold on the network as a whole. This translates to the spatial decision problem in indirect troubleshooting.

Formulation with indirect troubleshooting

Addressing the huge analysis space of concurrent events and distributed invariants with the implication problem of indirect troubleshooting is straightforward. First note that a common theme in both cases is divide-and-conquer: can we reduce a harder-to-verify task into to a collection of easier tasks? Denote the former task by γ, the latter by set $\sum = \{\sigma_1, \sigma_2 \cdots\}$, the goal is to find a systematic way to infer the more comprehensive γ out of the simpler $\sum$, i.e.$\sum \vdash \gamma$. We call the problem of deciding an invariant over all dynamics from invariants of concurrent events (e.g., Figure 1) the temporal decision problem; and the problem of deciding a network-wide invariant from invariants on several network partitions (e.g., Figure 2) the spatial decision problem. In the later case, we write $\mathsf{N} : \sum \vdash \mathsf{V} : \gamma$ to stress the disparate views (V, N) on different partitions.

3 A STRAWMAN SOLUTION

This section delves into potential solutions to the decision problem (i.e.$\sum \vdash \gamma$) within the existing paradigm that directly checks the relevant network elements. We quickly rule out system troubleshooting, as critical anomalies can often be latent and rare. For static verifiers, the decision problem requires extending their backend reasoning engines (e.g., satisfiability checking [6], model exploration [16]). Note that most reasoning engines handle the conclusion γ (e.g., blocking traffic on end-to-end connectivity) without a built-in mechanism to exploit the premises $\sum$ (if at all). That is, to obtain γ, existing engines often need to (smartly) generate and subsequently repeatedly analyze all network snapshots relevant to the premises (such as all possible link failure scenarios in Figure 1, all dataplane traces that might exhibit a security hole in Figure 2), potentially an explosive space for analysis. In fact, most optimization is focused on individual network incarnations, resulting from the complexity of networks (e.g., link failures), without inherently taking the complexity into account.

To address this issue, we propose a solution that incorporates network complexity into the model and utilizes it for reasoning. As a starting point, we consider using *fauré* [32] as a strawman solution for the temporal decision problem illustrated in Figure 1 (b). *Fauré* can naturally include the dynamics of the network into the model using the conditional table (or c-table) construct, which extends regular table with variables and an additional condition column that characterizes the presence of a data item to represent uncertain information. For example, `Fwd` (in Table 1) models all possible forwarding behaviors of Figure 1 (b) in a single table using additional boolean variables $\mathtt{l}_{12}$ ($\mathtt{l}_{23}$) that encode the status of the primary link $1-2$ ($2-3$). This model takes link dynamics into account and enables derivation of end-to-end reachability using a simple SQL query. The *fauré* evaluation strategy over c-tables takes care of link dynamics and gives the verification result in the form of query answers. For example, the different ACL entries in the answer `Reach` in Table 1 signal inconsistency, and the conditions explain why.

This strawman solution is a step forward from more traditional tools like Batfish [18]. Batfish treats link failures as an external environment, and explores all link failure snapshots only when provided as inputs. In each run on a specific input, the verifier is entirely unaware of the events that produced it. In contrast, *fauré* incorporates the uncertainty of concurrent events fully into the reasoning process (i.e. c-table evaluation), providing more robust reasoning support for concurrency. Despite these benefits, the strawman solution still has significant limitations. Firstly, while the c-table model facilities navigating the snapshots due to concurrency, it fails to reuse knowledge about each failure event, such as the fact that consistent backup link configuration is correct when only failure on the corresponding protected link is considered. More importantly, when generalized to the spatial decision problem, the strawman solution requires a universal data model (a c-table schema) as the basis of reasoning (querying), but the distributed invariants over distinct partitions can involve disparate views (schemas, or data models) of the network.

Fwd	N	X	A	Cond
	1	2	$\{1\}$	$\mathtt{l}_{12}$
	1	3	$\{1\}$	$\neg\mathtt{l}_{12}$
	2	3	$\{2\}$	$\mathtt{l}_{23}$
	2	4	$\{2\}$	$\neg\mathtt{l}_{23}$
	3	4	$\emptyset$	$\top$

Reach	S	D	A	Cond
	1	4	$\{1,2\}$	$\mathtt{l}_{12} \wedge \mathtt{l}_{23}$
	1	4	$\{1\}$	$\neg\mathtt{l}_{12}$
	1	4	$\{1,2\}$	$\mathtt{l}_{12} \wedge (\neg\mathtt{l}_{23})$

Table 1: `Fwd`: models all concurrent events in a single table by associating with each forwarding entry the triggering events; `Reach`: contains all the verification results, during the derivation of which the concurrent failures are fully processed within the reasoning engine (query processing).

Thus we argue for a paradigm shift, from direct to indirect verification of highly concurrent and distributed networks, for deciding logical implications among "component" invariants pertaining to interacting events and overlapping partitions.

4 THE CHASE

4.1 Modeling network invariants: network dependencies-by-example

Network invariants constrain certain aspects of the network. When the network state is modeled by a database (set of tables), network invariants become data dependencies. The dependencies can be general logical sentences or restricted (more familiar) forms like functional dependencies (key attributes in a table determine the rest of the columns) and

F	S	D	N	X
x_f	10.2	10.3	1	2
x_f	10.1	10.3	1	2

σ_1

F	S	D	N	X
x_f	10.2	10.3	1	2
y_f	10.1	10.3	1	2
$x_f = y_f$				

σ'_1

F	S	D	N	X
x_f	x_s	x_d		
x_f	y_s	y_d		
$x_s = y_s \wedge x_d = y_d$				

σ_D

Table 2: Dependencies for the network in Figure 2: σ_1 and σ'_1 give the semantics of rewrite at 1, σ_D describes destination based forwarding.

referential constraints (the presence of some tuples imply/require the existence of other rows). Our goal is to find an invariant representation to model and reason on networks.

To achieve this we adopt tableau [2, 4, 39], in which an invariant is provided through a possible example. The dependencies for the network in Figure 2 are shown as tableaux in Table 2. The network is modeled by a relation schema (F, S, D, N, X) describing the forwarding action for flow F with headers S (source) and D (destination), where the next hop at node N is X. σ_1 and σ'_1 describe the invariants implemented by the rewrite of node 1. σ_1 says that whenever a flow entry x_f for (10.2, 10.3) that matches the rewrite is observed at 1, the same flow (x_f) with a rewritten source (10.1) must be presented as well; σ'_1 adds that any two flows whose headers match the "before" and "after" sides of the rewriting rule must be considered as the same flow ($x_f = y_f$). Here, the mixed use of variables (x_f, y_f), constants, and equality gives an intuitive form of invariant description. A second example highlighting this strength is σ_D, which concisely expresses destination based forwarding: the headers (S, D attributes) are unchanged as long as they belong to the same flow x_f[2]. Significantly, the simple tableau form — describing invariants by a possible example — is also *complete*: σ_1 and σ'_1 correspond to two forms of dependencies, namely tuple generation dependency (tgd) and equality generation dependency (egd). Intuitively, tgd constrains what must be included and egd constrains what cannot be presented. It turns out that combining tgd and egd we can express arbitrary logical sentences [2] over a database model.

4.2 Solving the temporal decision problem: the chase at a glance

The temporal decision problem, $\sum \vdash \gamma$(where $\sum = \{\sigma_1, \sigma_2, \cdots\}$) over the same network model N (data scheme), can be decided by showing that any network instance I's (tables over N) that satisfy $\sum$ also satisfy γ. Our goal is to develop a test without referencing (explicitly constructing or exploring during the reasoning) such I's, and the chase [2–4, 39] is an elegant procedure that allows us to achieve this. To begin with, we can eliminate the "irrelevant" I's (i.e., those that do not satisfy $\sum$) indirectly by modifying γ itself. The next step is to verify whether modified γ is trivially true, as it has already ruled out any unsatisfying I's a priori.

[2]Irrelevant attributes are left empty, they can be filled with arbitrary distinct variables. We use the same scheme in the rest of the paper

	F	N	A
T_c	x_f	1	x_a
u_c	x_f	3	$x_a \cup \{a_1\}$

σ_{c12}

F	N	A
x_f	3	x_a
x_f	5	$x_a \cup \{a_2\}$

σ_{c34}

F	N	A
x_f	1	$\emptyset$
x_f	5	$\{a_1, a_2\}$

γ_c

Table 3: Dependencies in the temporal decision problem (Figure 1 (a)): $\sigma_{c12}, \sigma_{c34}$ and γ_c express consistent application of access control for failure event on link 1 − 2, 3 − 4, and all protected links in the entire network, respectively.

In the following we illustrate the chase procedure through the temporal decision problem of Figure 1 (a), showing how modifying one invariant to reflect a set of invariants is both straightforward and can be automated. Table 3 lists the invariants $\{\sigma_{c12}, \sigma_{c34}\}(= \sum)$, and$\gamma_c$, which express the consistency property of the network under link failures. These invariants apply to the schema Net(F, N, A), which represents that a flow F observed at location N has passed ACL A. σ_{c12} is the consistency property that a_1 is always applied to traffic x_f as it flows from 1 to 3. Similarly, σ_{c34} and γ_c are consistency expression for failure event on link 3 − 4 and all possible failures in the network respectively. To modify γ_c to account for $\sum$, we use the chase to successfully apply $\{\sigma_{c12}, \sigma_{c34}\}$ to γ_c (denoted as chase(γ_c, $\{\sigma_{c12}, \sigma_{c34}\}$)), as shown in Table 4. Chasing γ_c with σ_{c12} adds a new row $(x_f, 3, \{a_1\})$ since σ_{c12} asserts that $(x_f, 3, x_a \cup a_1)$ is also presented (in the database) whenever $(x_f, 1, x_a)$ is included. Similarly, chasing with c_{c34} adds $(x_f, 5, \{a_1, a_2\})$. The resulting γ'_c ($\langle T'_c, u_c\rangle$) is trivially true as $u_c \in T'_c$; thus we verify the temporal logical implication for the network in Figure 1 (a).

F	N	A
x_f	1	$\emptyset$
x_f	3	$\{a_1\}$
x_f	5	$\{a_1, a_2\}$

applying σ_{c12}

	F	N	A
	x_f	1	$\emptyset$
T'_c	x_f	3	$\{a_1\}$
	x_f	5	$\{a_1, a_2\}$
u_c	x_f	5	$\{a_1, a_2\}$

applying σ_{c34}

$\gamma'_c = \langle T'_c, u_c\rangle$

Table 4: Temporal decision test: chasing γ_c with $\sigma_{c12}, \sigma_{c34}$ yields γ'_c, which is trivially true; thus $\{\sigma_{c12}, \sigma_{c34}\} \vdash \gamma_c$. ■

4.3 Solving the spatial decision problem: addressing disparate network views

In the spatial decision problem N : $\sum \vdash$ V : γ, where $\sum, \gamma$ are over disparate "logical" network partitions (or views) that are modeled by different data schemas N, V. To accommodate this disparity, we make use of a mapping definition $E : N \to V$, a function from N to V to allow the application of $\sum$ to γ, in three steps: (1) Let $\gamma = \langle U, w\rangle$ which implies w in the presence of U, construct the inverse image of U, $E^{-1}(U)$. (2) Since $E^{-1}(U)$ is over N that corresponds to (or causes) U in γ, we chase it with $\sum$, obtaining chase($E^{-1}(U), \sum$); and then further "distill" the effects of $\sum$ on γ by re-applying the E, generating E(chase($E^{-1}(U), \sum$)). (3) Let $\gamma' = \langle E(\text{chase}(E^{-1}(U), \sum)), w\rangle$ (γ' is the result of "chasing" γ with $\sum$ when accounting for the schemas), like before, we complete the test by checking if γ' is trivially true.

V	F	N	X
u_1	x_f	10.2	10.3
u_2	y_f	10.1	10.4
w	z_f	10.2	10.4

$V : \gamma$
$(\gamma = \langle U, w\rangle,$
$U = \{u_1, u_2\})$

N	F	S	D	N	X
	x_f	x_s	x_d	x_s	1
T	x_f	y_s	y_d	1	2
	x_f	z_s	z_d	2	z_d
u	x_f			x_s	z_d

$E : N \longrightarrow V$

	F	S	D	N	X
t_1	x_f	10.2	x_d	10.2	1
t_2	x_f	y_s	y_d	1	2
t_3	x_f	z_s	10.3	2	10.3
t'_1	x'_f	10.1	x'_d	10.1	1
t'_2	x'_f	y'_s	y'_d	1	2
t'_3	x'_f	z'_s	10.4	2	10.4

$T_{u_1} \cup T_{u_2}$

Table 5: Compute $E^{-1}(\{u_1, u_2\})$**.**

Steps 1-2 are new, and we use Figure 2 to show them. The process is particularly straightforward when E is also expressed as a tableau — a form of "query-by-example" similar to the tableau dependencies: In the running example, $\sum = \{\sigma_1, \sigma'_1, \sigma_2, \sigma'_2, \sigma_D\}$ where $\sigma_1, \sigma'_1, \sigma_2, \sigma'_2, \sigma_D$ [3] are the local rewrite policy of 1, 2, and the destination based forwarding invariants that constrain the network's forwarding schema (F, S, D, N, X). $V : \gamma$ in Table 5 is over an end-to-end reachability view with schema (F, N, X) (flow F originated from N can reach X) and it describes a security hole, namely, if flows from 10.2 to 10.3 and 10.1 to 10.4 are successful, so are flows from 10.2 to 10.4.

Using E to compute the inverse image of γ on N: As shown in Table 5, E is a tableau query that maps the forwarding state (a table) with schema N to the end-to-end connectivity view with schema V. Intuitively, it describes forwarding from 1 to 2 in Figure 2 subject to arbitrary rewriting. Observe that with the pattern matching implicit in this "query-by-example", the sending hosts must use their own address as the source (x_s) and the next hop is constrained by the topology. To compute the inverse image of U, we only need to find the "examples" (the tableau T when considered as an instance) producing u_1, u_2 and take their union, as shown in $T_{u_1} \cup T_{u_2}$ in Table 5, i.e. for each u_1 (u_2) in γ, we find a substitution θ that equals it to u of E, $T_{u_1}(T_{u_2}) = \theta(T)$. Let $Z = T_{u_1} \cup T_{u_2} = E^{-1}(U)$.

	S	D
t_1	10.2	10.3
t_2	10.2	10.3
t_3	10.2	10.3
t'_1	10.1	10.4
t'_2	10.1	10.4
t'_3	10.1	10.4

applying σ_D

	F	S	D	N	X
t_1					
t_2	x_f	10.2	10.3	1	2
v_1	x_f	10.1	10.3	1	2
t_3					
t'_1					
t'_2					
t'_3					

applying σ_1

	F	S	D	N	X
t_1	x_f			10.2	1
t_2					
v_1	x_f	10.1	10.3	1	2
t_3					
t'_1	x_f				
t'_2	x_f	10.1	10.4	1	2
t'_3	x_f			2	10.4

applying σ_2

Table 6: Chasing Z with $\sum$: only updated entries are shown.

Computing the effects of $N : \sum$ relevant to $V : \gamma$ We first chase the inverse image Z as shown in Table 6, and then re-apply E (to extract the relevant effects). The chase applications to Z (Table 6) include egd dependencies σ'_1, σ'_2, and σ_D. Unlike the application of tgds like σ_1 and σ_2 that insert a new tuple v_1 with respect to t_2 (corresponding the forwarding entry associated with the new packet after rewriting), egd "resets" the value of entries to satisfy the corresponding dependency. Chasing with σ_D sets the S and D fields of the same flow to the same value; and chasing with σ'_1 and σ'_2 set the flow ids whose S, D fields match the "before" and "after" fields of a rewrite rule to be the same. The result chase(Z, $\sum$), after re-applying E, gives us $\gamma'(= E(\text{chase}(Z, \sum)))$ (details omitted, due to limited space) which, following the same process as in the temporal decision problem, is trivially true (i.e. $w \in \gamma'$).

[3] see § 4.1 for details on $\sigma_1, \sigma'_1, \sigma_D$; σ_2 and σ'_2 are similar to σ_1 and σ'_1 respectively.

Theorem 4.1. *The spatial decision problem (and its solution) subsumes the temporal decision problem (and its solution)*

When V = N, the spatial decision problem $N : \sum \vdash V : \gamma$ degenerates to the temporal version $\sum \vdash \gamma$, and the spatial decision test of $\langle E(\text{chase}(E^{-1}(\{u_1, u_2\}), \sum)), w\rangle$ reduces to chase$(\gamma, \sum)$.

Previously, the problems of dependency implication and that of the interaction between views and dependencies were studied separately [2]. To our best knowledge, we are the first to show that the later subsumes the former. This allows us to address the decision problem with concurrency in networking as a special case of that with distribution.

5 TOWARDS A FLEXIBLE FRAMEWORK

The tests that we developed for the spatial decision problems allow us to run chase over disparate network views using the inverse image computation, denoted $E^{-1}(U)$, where $E : N_1 \longrightarrow N_2$, and $\gamma = \{U, w\}$ is defined over N_1, that traces the dependency $N_1 : \gamma$ to a different data model N_2. If we abuse the notation a bit, using chase(σ_1, σ_2) to denote the impact of $V_2 : \sigma_2$ on $V_1 : \sigma_1$ across two schemas $V_{1,2}$, where $E_i : N \longrightarrow V_i$, $i = 1, 2$. (i.e.chase(σ_1, σ_2) as a loose shorthand for $E_1(\text{chase}(E_1^{-1}(\sigma_1), E_2^{-1}(\sigma_2))))$, we obtain an extended chase notion that accounts for different data schemas. This allows natural formulation of a wide range of troubleshooting problems, solving issues with the two dark clouds.

Specifically, we envision a framework with the chase that can play an important role in managing network failures and evolutions. Figure 3 illustrates this potential: in the two administrative domains (their schemes being V_1, V_2 w.r.t a common model N), each is maintaining a set of invariants (policies) — σ's in domain 1 and γ's in domain 2 — with the help of some verifiers, and simultaneously checking for adhoc errors — $\epsilon_{1,2}$. In particular, the admin of domain 1 with restricted visibility can use the chase to answer questions shown in Table 7. It's important to emphasize that our framework is highly adaptable and can work with a wide range of existing tools. For instance, when it comes to identifying individual failures on a real network through system troubleshooting, our framework can provide valuable insights by answering critical questions such as, "Based on the detected failures, are there any other potential failures that could be more severe? And are there any important network properties that remain unaffected?" Similarly, for analyzing complex, large-scale networks using static reasoning, our framework can effectively integrate the results obtained from traditional tools used for smaller network partitions or

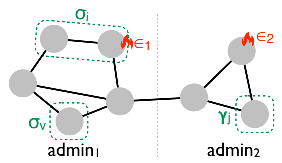


Figure 3: Example: verified invariants, detected errors.

scenario	decision problem	test
evolution assessment	is new policy σ_v redundant (impossible)?	$\texttt{chase}(\sigma_v, \sum) = \top\ (\bot)$?
symptom diagnosis	`admin1`: is remote ϵ_2 bound to occur if ϵ_1 is detected?	$\texttt{chase}(\epsilon_2, \{\epsilon_1, \sum\}) = \top$?
	is detected failure ϵ_1 a sure symptom of $\sigma_\texttt{i}$?	$\texttt{chase}(\sigma_\texttt{i}, \{\epsilon_1, \sum\}) = \bot$?
impact analysis	is $\sigma_\texttt{i}$ unaffected when a new error ϵ_1 is detected?	$\texttt{chase}(\sigma_\texttt{i}, \{\epsilon_1, \sum\}) = \sigma_\texttt{i}$?

Table 7: Example questions (decision problems) that the chase can answer to guide the admin of domain 1 in her interaction with existing system troubleshooting and static verification tools. Here, $\sum = \{\sigma_1, \cdots, \sigma_\texttt{i}, \cdots\} \cup \{\cdots \gamma \cdots\}$ is the set of all invariants (verified with existing tools).

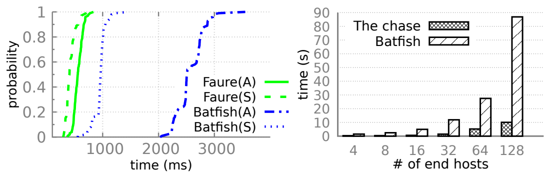
lower levels of concurrency. This way, we can obtain a more comprehensive understanding of the network and its behavior, leading to better-informed decisions and more efficient troubleshooting.

6 PRELIMINARY RESULTS

We use early stage experiments to (1) quantify how prior tools like Batfish [17] deteriorate while the strawman solution *fauré* [32] gives some relief as the concurrency (of events) increase on a link failure example like Figure 1; and (2) quantify the overhead and advantage of indirect troubleshooting with distributed invariants on a firewall example near Figure 2. In both experiments, we embed the toy examples into an ISP topology (Rocketfuel dataset [44]), using an Intel(R) Core(TM) i7-4790 machine with 25 GB of memory.

The cost of concurrent events with direct verification: In Figure 4a, Batfish (S) and *Fauré* (S) show the running time of verifying a **s**ingular link failure event, Batfish (A) and *Fauré* (A) show the running time for **a**ll possible (link failures) events in the entire network. Batfish is close to *fauré* in performance verifying a singular event, but quickly slows down and is out-paced by *fauré* when all four failure events in Figure 4a are considered, as the latter inherently models concurrency in its reasoning process. We also note that, the temporal decision test (§ 4.2) using chase on the same setup only takes (on average) 6 ms, making the chase an attractive tool to "compose" Batfish/*fauré* verification results on a singular event. When those single-event verification are parallelized, the chase, with negligible overhead, can drive down the verification time of the entire network to that of verifying a single event.

The strength for distributed invariants with indirect troubleshooting via the chase: Figure 4b depicts the running time of verifying distributed invariants on varying number of hosts in Figure 2, where the attacks on the distributed firewalls may be launched. Batfish has to check flows between pairs of source and destination under user's manual guidance. The chase, on the other hand, takes the distributed invariants as dependencies, and automatically decides the security hole as captured in the conclusion dependency in the spatial decision test (§ 4.3). Our naive implementation of the chase consistently outperforms the highly-optimized Batfish on all inputs. As the number of hosts increase, the number of unique flows and consequently the time taken by Batfish to exhaustively search the trace of the attack increases. While the chase exhibits a similar slow-down, we stress that the chase looks for the security hole by modifying



(a) **(b)**

Figure 4: (a) Impact of concurrency on Batfish and *fauré*; (b) Comparing the chase and Batfish for distributed firewalls

the conclusion using all data dependencies in the network automatically instead of considering each source and destination pair one-by-one.

7 RELATED WORK

Indirect troubleshooting is inspired and is designed to complement prior troubleshooting tools. This section only outlines our use of chase as a step forward in managing networked systems by database techniques: Distributed data query as a programming language for network protocols was studied in declarative networking [13, 14, 33–35, 37, 40] where a datalog-like language produces extremely concise code (Chord [45] in 47 rules [36]). The ability of datalog to separate policy from distributed state management was noted in software-defined networking [10, 15, 23, 30, 31] and network verification [18, 38, 51]. We take a step forward in exploiting data management: we formula network troubleshooting as a general decision problem that can be solved with the novel chase procedure [2, 4, 39].

8 CONCLUSION

We introduce the chase, a remarkable process for deciding implications among network invariants, and propose it as a new verification primitive. By systematically composing simpler invariants and inferring relations among parts of the network, the chase complements more direct approach — the current paradigm of verifying an invariant in the entire network. Together, existing tools and the chase may allow us to address several verification challenges — such as the complexity of distributed system, the decision support needed for evolution — that are hardly seen in other domains. As computer networks have been and will likely remain an unparalleled — ever-evolving, distributed, highly dynamic — infrastructure, it is important to develop full verification support; and this paper reports our first step in this direction with the chase.

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)*.

[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] A. V. Aho, C. Beeri, and J. D. Ullman. 1979. The Theory of Joins in Relational Databases. *ACM Trans. Database Syst.* 4, 3 (sep 1979), 297–314. https://doi.org/10.1145/320083.320091

[4] Alfred V. Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. 1979. Equivalences among relational expressions. *SIAM J. Comput.* 8, 2 (1979), 218–246.

[5] Paramvir Bahl, Ranveer Chandra, Albert Greenberg, Srikanth Kandula, David A. Maltz, and Ming Zhang. 2007. Towards Highly Reliable Enterprise Network Services via Inference of Multi-Level Dependencies. In *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '07)*. Association for Computing Machinery, New York, NY, USA, 13–24. https://doi.org/10.1145/1282380.1282383

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

[7] 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, 476–489. https://doi.org/10.1145/3230543.3230583

[8] Ryan Beckett and Ratul Mahajan. 2019. Putting Network Verification to Good Use. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks (HotNets '19)*. Association for Computing Machinery. https://doi.org/10.1145/3365609.3365866

[9] Ryan Beckett and Ratul Mahajan. 2020. A General Framework for Compositional Network Modeling. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks (HotNets '20)*. Association for Computing Machinery, New York, NY, USA, 8–15. https://doi.org/10.1145/3422604.3425930

[10] 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

[11] Mike Chen, Emre Kiciman, Anthony Accardi, Armando Fox, and Eric Brewer. 2003. Using Runtime Paths for Macroanalysis. In *Proceedings of the 9th Conference on Hot Topics in Operating Systems - Volume 9 (HOTOS'03)*. USENIX Association, USA, 14.

[12] Mike Y. Chen, Anthony Accardi, and Dave Patterson. 2004. Path-Based Failure and Evolution Management. In *First Symposium on Networked Systems Design and Implementation (NSDI 04)*. USENIX Association, San Francisco, CA. https://www.usenix.org/conference/nsdi-04/path-based-failure-and-evolution-management

[13] 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

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

[15] 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

[16] 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 *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, USA, 217–232.

[17] 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, 469–483.

[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 *(NSDI'15)*. USENIX Association, USA.

[19] Rodrigo Fonseca, George Porter, Randy H. Katz, and Scott Shenker. 2007. X-Trace: A Pervasive Network Tracing Framework. In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07)*. USENIX Association, Cambridge, MA. https://www.usenix.org/conference/nsdi-07/x-trace-pervasive-network-tracing-framework

[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, 300–313. https://doi.org/10.1145/2934872.2934876

[21] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Mazières, and Nick McKeown. 2014. I Know What Your Packet Did Last Hop: Using Packet Histories to Troubleshoot Networks. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, USA, 71–85.

[22] Brandon Heller, Colin Scott, Nick McKeown, Scott Shenker, Andreas Wundsam, Hongyi Zeng, Sam Whitlock, Vimalkumar Jeyakumar, Nikhil Handigol, James McCauley, Kyriakos Zarifis, and Peyman Kazemian. 2013. Leveraging SDN Layering to Systematically Troubleshoot Networks. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN '13)*. ACM, New York, NY, USA, 37–42. https://doi.org/10.1145/2491185.2491197

[23] 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

[24] 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, 200–213. https://doi.org/10.1145/3341302.3342094

[25] Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Paramvir Bahl. 2009. Detailed Diagnosis in Enterprise Networks. *SIGCOMM Comput. Commun. Rev.* 39, 4 (aug 2009), 243–254. https://doi.org/10.1145/1594977.1592597

[26] 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, 99–112.

[27] 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.

[28] 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, 49–54. https://doi.org/10.1145/2342441.2342452

[29] Ramana Rao Kompella, Jennifer Yates, Albert Greenberg, and Alex C. Snoeren. 2010. Fault Localization via Risk Modeling. *IEEE Transactions on Dependable and Secure Computing* 7, 4 (2010), 396–409. https://doi.org/10.1109/TDSC.2009.37

[30] 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

[31] 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 *(OSDI'10)*.

[32] Fangping Lan, Bin Gui, and Anduo Wang. [n. d.]. Faure: a partial approach to network analysis *(ACM Workshop on Hot Topics in Networks (HotNets), November, 2021)*.

[33] 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

[34] 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

[35] 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 *(SIGMOD '06)*. ACM. https://doi.org/10.1145/1142473.1142485

[36] Boon Thau Loo, Tyson Condie, Joseph M. Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. 2005. Implementing Declarative Overlays. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP '05)*. ACM, New York, NY, USA, 75–90. https://doi.org/10.1145/1095810.1095818

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

[38] 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

[39] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. 1979. Testing Implications of Data Dependencies. *ACM Trans. Database Syst.* 4, 4 (dec 1979), 455–469. https://doi.org/10.1145/320107.320115

[40] 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

[41] 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, 69–83. https://doi.org/10.1145/2837614.2837657

[42] 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

[43] Colin Scott, Andreas Wundsam, Barath Raghavan, Aurojit Panda, Andrew Or, Jefferson Lai, Eugene Huang, Zhi Liu, Ahmed El-Hassany, Sam Whitlock, H.B. Acharya, Kyriakos Zarifis, and Scott Shenker. 2014. Troubleshooting Blackbox SDN Control Software with Minimal Causal Sequences. *SIGCOMM Comput. Commun. Rev.* 44, 4 (aug 2014), 395–406. https://doi.org/10.1145/2740070.2626304

[44] Neil Spring, Ratul Mahajan, and Thomas Anderson. [n. d.]. Rocketfuel: An ISP Topology Mapping Engine. http://research.cs.washington.edu/networking/rocketfuel/. ([n. d.]).

[45] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. 2003. Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Trans. Netw.* 11, 1 (feb 2003), 17–32. https://doi.org/10.1109/TNET.2002.808407

[46] 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*. https://doi.org/10.1109/INFCOM.2005.1498492

[47] Xieyang Xu, Ryan Beckett, Karthick Jayaraman, Ratul Mahajan, and David Walker. 2021. Test Coverage Metrics for the Network. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 775–787. https://doi.org/10.1145/3452296.3472941

[48] 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, 599–614. https://doi.org/10.1145/3387514.3406217

[49] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Automatic Test Packet Generation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '12)*. Association for Computing Machinery, New York, NY, USA, 241–252. https://doi.org/10.1145/2413176.2413205

[50] Peng Zhang, Aaron Gember-Jacobson, Yueshang Zuo, Yuhao Huang, Xu Liu, and Hao Li. 2022. Differential Network Analysis. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 601–615. https://www.usenix.org/conference/nsdi22/presentation/zhang-peng

[51] 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, 81–87. https://doi.org/10.1145/3422604.3425936

[52] 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