ACORN: Network Control Plane Abstraction using Route Nondeterminism

Divya Raghunathan
Princeton University
Princeton, USA
dr31@cs.princeton.edu

Ryan Beckett
Microsoft Research
Redmond, USA
ryan.beckett@microsoft.com

Aarti Gupta
Princeton University
Princeton, USA
aartig@cs.princeton.edu

David Walker
Princeton University
Princeton, USA
dpw@cs.princeton.edu

***Abstract*—Networks are hard to configure correctly, and misconfigurations occur frequently, leading to outages or security breaches. Formal verification techniques have been applied to guarantee the correctness of network configurations, thereby improving network reliability. This work addresses verification of distributed network control planes, with two distinct contributions to improve the scalability of verification. Our first contribution is a hierarchy of abstractions of varying precision which introduce nondeterminism into the procedure that routers use to select the best available route. We prove the soundness of these abstractions and show their benefits. Our second contribution is a novel SMT encoding which uses symbolic graphs to encode all possible stable routing trees that are compliant with the given network control plane configurations. We have implemented our abstractions and SMT encoding in a prototype tool called ACORN. Our evaluations show that our abstractions can provide significant relative speedups (up to 323x) in performance, and ACORN can scale up to $\approx 37,000$ routers in data center benchmarks (with FatTree topologies, running shortest-path routing and valley-free policies) for verifying reachability. This far exceeds the performance of existing control plane verifiers.**

I. Introduction

Bugs in configuring networks can lead to expensive outages or critical security breaches, and misconfigurations occur frequently [1], [2], [3], [4], [5], [6]. Thus, there has been great interest in formal verification of computer network configurations. Many initial efforts targeted the network *data plane*, *i.e.,* the forwarding rules in each router that determine how a given packet is forwarded to a destination. Many of these methods have been successfully applied in large data centers in practice [7], [8], [9]. In comparison, formal verification of the network *control plane* is more challenging.

Traditional control planes use distributed protocols such as OSPF, BGP, and RIP [10] to compute a network data plane based on the route announcements received from peer networks, the current failures detected, and the router configurations. In control plane verification, one must check that *all* data planes that emerge due to the router configurations are correct. There has been much recent progress in control plane verification. Fully symbolic SMT-based verifiers [11], [12], [13] usually work well for small-sized networks, but have not been shown to scale to medium-to-large networks. Simulation-based verifiers [14], [15], [16], [13], [17], [18] scale better, but in general, do not provide full symbolic reasoning, *e.g.,* for considering all external route announcements. Our work is motivated by this gap: we aim to provide full symbolic reasoning *and* improve the scalability of verification. We address this challenge with two main contributions – a novel hierarchy of control plane abstractions, and a new symbolic graph-based SMT encoding for control plane verification.

Hierarchy of nondeterministic abstractions. Our novel control plane abstractions introduce nondeterminism in the procedure that routers use to select a route – we call these the *Nondeterministic Routing Choice (NRC) abstractions*. Instead of forcing a router to pick the *best* available route, we allow it to *nondeterministically choose* a route from a subset of available routes which includes the best route. The number of non-best routes in this set determines the precision of the abstraction; our least precise abstraction corresponds to picking *any* available route that is compliant with policy.

Our main insight here is that determining the best route may not be needed for verification of many correctness properties that network operators care about, such as reachability (*e.g.,* when the number of hops may not matter), valley-freedom, or no-transit (Gao-Rexford conditions [19]). On the other hand, for policy-based routing, it is still important to model other protocol features such as route filters. Our results show, for the first time, that nondeterministic routing abstractions can successfully verify such properties and provide significant gains in performance and scalability. Although some other efforts [12], [20] have also proposed to abstract the decision process in BGP (details in §VII), we elucidate and study the general principle for generic distributed protocols, prove it sound, and reveal a range of precision-cost tradeoffs.

The potential downside of considering non-best routes is that our abstractions may lead to false positives, *i.e.,* we could report property violations although the *best* route may actually satisfy the property. In such cases, we propose using a more precise abstraction that models more of the route selection procedure. Our experiments (§VI) demonstrate that the NRC abstractions can successfully verify a wide range of networks and common policies and offer significant performance and scalability benefits in symbolic SMT-based verification. Although our abstractions are sound for verification of specified failures (§IV), we focus on verification without failures here, and plan to consider failures in future work.

Symbolic graph-based SMT encoding. Our novel SMT

https://doi.org/

encoding uses *symbolic graphs* [21] (where a Boolean variable is associated with each edge in the network topology) to model the stable states of a network control plane. Our encoding can leverage specialized SMT solvers such as MonoSAT [21] that provide support for graph-based reasoning, as well as standard SMT solvers such as Z3 [22].

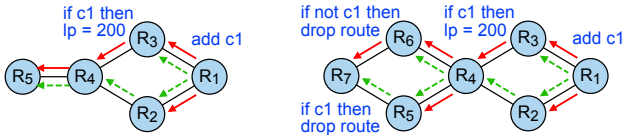
Experimental evaluation. We have implemented our NRC abstractions and symbolic graph-based SMT encoding in a prototype tool called ACORN (**A**bstracting the **CO**ntrol plane using **R**oute **N**ondeterminism). We present a detailed evaluation on benchmark examples that include synthetic data center examples with FatTree topologies [23], as well as real topologies from Topology Zoo [24] and BGPStream [25] running well-known network policies, where we verify reachability and other properties of interest. All benchmark examples are successfully verified using an NRC abstraction (96% of examples with our least precise abstraction, and the remaining 4% using a more precise abstraction). These benchmarks, including some new examples that we created, are publicly available [26]. ACORN could verify reachability in large FatTree benchmarks with about 37,000 nodes (running common policies) within an hour. This kind of scalability is needed in modern data centers with tens of thousands of routers that run distributed routing protocols such as BGP [27]. We compared ACORN with two publicly available state-of-the-art control plane verifiers on the data center benchmarks, and our results show that our tool scales an order of magnitude better.

To summarize, we make the following contributions:

1) We present a hierarchy of novel control plane abstractions, called the NRC abstractions, that add nondeterminism to a general route selection procedure (§IV). We prove our abstractions sound and empirically show that they enable a precision-cost tradeoff in verification. Although our focus is on SMT-based verification, these abstractions could be used with other methods as well.
2) We present a novel SMT encoding (§V) (based on symbolic graphs [21]) to capture distributed control plane behavior. This leverages SMT solvers that support graph-based reasoning, as well as standard SMT solvers.
3) We implemented our abstractions and SMT encoding in a prototype tool called ACORN and present a detailed evaluation (§VI) on synthetic data center benchmarks and real-world topologies with well-known network policies.

II. Motivating Examples

In a distributed routing protocol, routers exchange *route announcements* containing information on how to reach various destinations. On receiving a route announcement, a router updates its internal state and sends a route announcement to neighboring routers after processing it as per the routing configurations. In well-behaved networks, this distributed decision process converges to a *stable state* [28] in which the internal routing information of each router does not change upon receiving additional route announcements. The best route selected by each router defines a *routing tree*: if router u selects the route announcement sent by router v for destination d, then u will forward data packets with destination d to v.



(a) Example 1 (b) Example 2

Fig. 1: Examples showing correct verification result with an NRC abstraction. Red arrows show the routing tree in the real network, and green arrows show an additional routing tree allowed in the abstraction.

Example 1 (Motivating example)**.** Consider the network in Figure 1a (from ShapeShifter [16]) with five routers running the Border Gateway Protocol (BGP), described in Appendix A, where actions taken by routers are shown along the edges.

The verification task is to check whether routes announced at R_1 can reach R_5. The network uses the BGP community attribute, a list of string tags, to ensure that R_4 prefers to route through R_3: the community tag $c1$ is added along the edge (R_1, R_3), which causes the local preference (lp) to be set to 200 along the edge (R_3, R_4). Routes with higher local preference are preferred (the default local preference is 100). Thus, the best route at R_4 is through R_3 and the corresponding routing tree is shown by red (solid) arrows.

Note that R_5 can receive a route even if R_4 chooses to route through R_2 instead, though this route is not the best for R_4. Thus, R_5 can reach the destination regardless of the choice R_4 makes. This observation captures the basic idea in our NRC abstractions– intuitively, we explore *multiple* available routes at a node: the best route as well as other routes. Then we check if R_5 receives a route under *each of these possibilities*. Since R_5 can reach R_1 in all routing trees considered by our abstraction we correctly conclude that it can reach R_1.

False positives and refinement. The NRC abstractions are sound, *i.e.,* when verification with an abstraction is successful, the property is guaranteed to hold in the network. However, verification with an abstraction could report a false positive, *i.e.,* a property violation even when the network satisfies the property. In Figure 1a, suppose R_5 drops routes without the tag $c1$. In the real network, R_5 will receive a route, since the route sent by R_4 has the tag $c1$. However, verification with an abstraction that considers all possible routes would report that R_5 *cannot* reach the destination, with a counterexample where R_4 routes through R_2 and its route announcement is dropped by R_5. Here, an NRC abstraction higher up in the precision hierarchy, *e.g.,* one which chooses a route with maximum local preference but abstracts the path length, will verify that R_5 receives a route, thereby eliminating the false positive.

Path-sensitive reasoning. Even our least precise abstraction can verify many interesting policies due to our symbolic SMT-based approach which tracks correlations between choices made at different routers, which other tools [16] do not track.

SRP instance: $SRP = (G, A, a_d, \prec, \text{trans}),\ G = (V, E, d)$
SRP solution: $\mathcal{L} : V \to A_\infty$

$$\mathcal{L}(u) = \begin{cases} a_d & \text{if } u = d \\ \infty & \text{if } \text{attrs}_{\mathcal{L}}(u) = \emptyset \\ a \in \text{attrs}_{\mathcal{L}}(u), \text{ minimal by } \prec & \text{if } \text{attrs}_{\mathcal{L}}(u) \neq \emptyset \end{cases}$$

$$\text{attrs}_{\mathcal{L}}(u) = \{a \mid (e, a) \in \text{choices}_{\mathcal{L}}(u)\}$$

$$\text{choices}_{\mathcal{L}}(u) = \{(e, a) \mid e = (v, u), \\ a = \text{trans}(e, \mathcal{L}(v)),\ a \neq \infty)\}$$

Fig. 2: Cheat sheet for SRP [30].

Example 2 (Path-sensitivity)**.** Figure 1b shows another BGP network (from Propane [29]), with seven routers and destination R_1. We would like to verify that R_7 can reach R_1.

In the real network, R_4 chooses the route from R_3 which has higher local preference (as shown by red/solid arrows). Under the least precise NRC abstraction, R_4 could choose the route from R_2 instead. Regardless of R_4's choice, the community tags in the routes received by R_5 and R_6 are the same, and so R_7 will receive a route either way – our abstraction tracks this correlation and correctly concludes that R_7 can reach R_1.

III. Preliminaries

In this section we briefly cover the background on the key building blocks required to describe our technical contributions. Our NRC abstractions are formalized using the Stable Routing Problem (SRP) model [30], [13], a formal model of network routing for distributed routing protocols. We also briefly describe SMT-based verification using the SRP model (*e.g.,* Minesweeper [11]) and support for graph-based reasoning in the SMT solver MonoSAT [21].

Definition 1 (Stable Routing Problem (SRP) [30])**.** An SRP is a tuple $(G,\ A,\ a_d,\ \prec,\ \text{trans})$ where $G = (V,\ E,\ d)$ is a graph representing the network topology with vertices V, directed edges E, and destination d; A is a set of *attributes* representing route announcements; $a_d \in A$ denotes the initial route sent by d; $\prec\ \subseteq A \times A$ is a partial order that models the route selection procedure (if $a_1 \prec a_2$ then a_1 is preferred); $\text{trans} : E \times A_\infty \to A_\infty$, where $A_\infty = A \cup \{\infty\}$ and ∞ denotes no route, is a *transfer function* that models the processing of route announcements sent from one router to another.

Figure 2 summarizes the important notions for the SRP model [30]. The main difference from routing algebras [31], [32] is that the SRP model includes a network topology graph G to reason about a given network and its configurations.

SRP solutions. A solution of an SRP is a labeling function $\mathcal{L} : V \to A_\infty$ which represents the final route (attribute) chosen by each node when the protocol converges. An SRP can have multiple solutions, or it may have none. Any SRP solution satisfies a *local stability condition*: each node selects the best among the route announcements received from its neighbors.

Example 3 (SRP example)**.** The network in Figure 1a running a simplified version of BGP (simplified for pedagogic reasons) is modeled using an SRP in which attributes are tuples comprising an integer (local preference), a set of bit vectors (community tags), and a list of vertices (the path). We use $a.lp$, $a.comms$, and $a.path$ to refer to the elements of an attribute a. The initial attribute at the destination, $a_d = (100, \emptyset, [\,])$. The preference relation $\prec$ models the BGP route selection procedure which is used to select the best route. The attribute with highest local preference is preferred; to break ties, the attribute with minimum path length is preferred (more details are in Appendix A). The transfer function for edge (R_1, R_3) adds the tag $c1$ and prepends R_1 to the path, returning $(100, a.comms \cup \{c1\}, [R_1] + a.path)$. The transfer function for edge (R_3, R_4) sets the local preference to 200 if the tag $c1$ is present, *i.e.,* if $c1 \in a.comms$ it returns $(200, a.comms, [R_3] + a.path)$; otherwise, it returns $(100, a.comms, [R_3] + a.path)$. The transfer function for other edges (u, v) prepends u to the path, sets the local preference to the default value (100), and propagates the community tags.

SMT-based verification using SRP. Minesweeper [11] encodes the SRP instance for the network using an SMT formula N, such that satisfying assignments of N correspond to SRP solutions. To verify if a property encoded as a formula P holds, the satisfiability of $F = N \wedge \neg P$ is checked. If F is satisfiable, a property violation is reported. Otherwise, the property holds over the network (assuming N is satisfiable; otherwise there are no stable paths).

SMT with theory solver for graphs. MonoSAT [21] is an SMT solver with support for *monotonic* predicates. A predicate p is (positive) monotonic in a variable u if whenever $p(\ldots u = 0 \ldots)$ is true, $p(\ldots u = 1 \ldots)$ is also true. Graph reachability is a monotonic predicate: if node v_1 can reach node v_2 in a graph with an edge removed, it can still reach v_2 when the edge is added. MonoSAT leverages predicate monotonicity to provide efficient theory support for graph-based reasoning using a *symbolic graph*, a graph with a Boolean variable per edge. Formulas can include these Boolean edge variables as well as monotonic predicates such as reachability and max-flow. MonoSAT has been used to check reachability in data planes in AWS networks [33], [34], but not in *control planes*, as we do in this work.

IV. NRC Abstractions

We formalize our NRC abstractions as *abstract* SRP instances, which are parameterized by a partial order.

Definition 2 (Abstract SRP)**.** For an SRP $S = (G, A, a_d, \prec, trans)$, an abstract SRP $\widehat{S}_{\prec'}$ is a tuple $(G, A, a_d, \prec', trans)$, where G, A, a_d, and $trans$ are defined as in the SRP S, and $\prec' \subseteq A_\infty \times A_\infty$ is a partial order which satisfies

$$\forall B \subseteq A,\ \text{minimal}(B, \prec) \subseteq \text{minimal}(B, \prec') \qquad (1)$$

where $\text{minimal}(B, \prec) = \{a \in B \mid \nexists a' \in B.\ a' \neq a \wedge a' \prec a\}$ denotes the set of minimal elements of B according to $\prec$. Condition (1) specifies that for any set of attributes B, the minimal elements of B by $\prec$ are also minimal by $\prec'$.

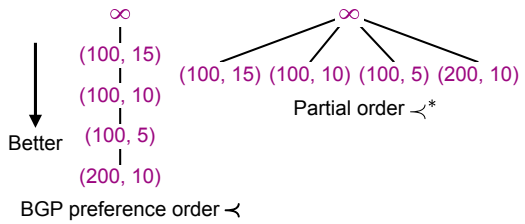


Fig. 3: Partial orders in concrete and abstract SRPs.

Note that condition (1) ensures that the solutions (*i.e.,* minimal elements) at any node in an SRP are also solutions at the same node in the abstract SRP, *i.e.,* the NRC abstractions over-approximate the behavior of an SRP. The precision of an NRC abstraction depends on the partial order used. Our least precise abstraction uses $\prec^*$, in which any two attributes are incomparable and ∞ is worse than all attributes, and corresponds to choosing any available route. The following example illustrates solutions of an abstract SRP $\widehat{S}_{\prec^*}$.

Example 4 (Abstract SRP $\widehat{S}_{\prec^*}$)**.** Figure 3 shows Hasse diagrams for partially ordered sets comprising simplified BGP attributes (pairs with local preference and path length; ∞ denotes no route) at a node u and two partial orders: (1) $\prec$, the partial order in the standard (concrete) SRP (lifted to A_∞) that models BGP's route selection procedure (shown on the left), and (2) $\prec^*$, the partial order corresponding to choosing any available route (shown on the right). Attributes appearing lower in the Hasse diagram are considered better. Hence, in the concrete SRP, u will select $(200, 10)$. In the abstract SRP, any element that is minimal by $\prec^*$ can be a solution for u so u nondeterministically selects an available route. Observe that $(200, 10)$, the solution for u in the concrete SRP, is *guaranteed* to be one of the solutions for u in the abstract SRP. This over-approximation due to condition (1) ensures that our abstraction is sound, *i.e.,* it will not miss any property violations.

Verification with an NRC abstraction. To verify that a property holds in a network using an abstraction $\prec'$, we construct an SMT formula $\widehat{N}$ such that satisfying assignments of $\widehat{N}$ are solutions of the abstract SRP $\widehat{S}_{\prec'}$ for the network, and conjoin it with the negation of an encoding of the property P to get a formula $F = \widehat{N} \wedge \neg P$. If F is unsatisfiable, all solutions of $\widehat{S}_{\prec'}$ satisfy the property and verification is successful. Otherwise, we report a violation with a counterexample (a satisfying assignment), and a user can perform refinement (described later in this section). Our approach is sound for properties that hold for *all* stable states, *i.e.,* properties of the form $\forall \mathcal{L} \in Sol(S).P(\mathcal{L})$, where $Sol(S)$ denotes the SRP solutions for the network. Like Minesweeper [11], our approach only models the stable states of a network and cannot verify properties over transient states that arise before convergence.

Lemma 1. [Over-approximation] For an SRP S and corresponding abstract SRP $\widehat{S}_{\prec'}$ with solutions $Sol(S)$ and $Sol(\widehat{S}_{\prec'})$ respectively, $Sol(S) \subseteq Sol(\widehat{S}_{\prec'})$.

Protocol	Partial order	Best route
OSPF	$\prec^*$	Any
	$\prec_{(pathcost)}$	min path cost
	$\prec_{ospf}$	min path cost, min router ID
BGP	$\prec^*$	Any
	$\prec_{(lp)}$	max lp (local preference)
	$\prec_{(lp,pl)}$	max lp, min path length
	$\prec_{(lp,pl,MED)}$	max lp, min path length, min MED (Multi-exit Discriminator)
	$\prec_{bgp}$	max lp, min path length, min MED, min router ID

Fig. 4: Hierarchy of NRC abstractions for OSPF and BGP.

The proof follows from the definition of SRP solutions and the over-approximation condition (1) (full proof in Appendix B).

Theorem 1. [Soundness] Given SMT formulas $\widehat{N}$ and N modeling the abstract and concrete SRPs respectively and SMT formula P encoding the property to be verified, if $\widehat{N} \wedge \neg P$ is unsatisfiable, then $N \wedge \neg P$ is also unsatisfiable.

The proof follows from Lemma 1 and is shown in Appendix B.

Verification under failures. We model link failures using ∞, which denotes no route (device failures are modeled as failures of all incident links). Let F denote a set of failed links. Given SRP $S = (G, A, a_d, \prec, \text{trans})$, we model network behavior under failures F using an SRP $S_F = (G, A, a_d, \prec, \text{trans}_F)$ where trans_F returns ∞ along edges in F and is the same as trans for other edges. We similarly define an abstract SRP for S_F, $\widehat{S}_{\prec' F} = (G, A, a_d, \prec', \text{trans}_F)$; it only differs from S_F in the partial order $\prec'$. Since Lemma 1 holds for an arbitrary concrete SRP S, it holds for S_F, *i.e.,* any solution of S_F is also a solution of $\widehat{S}_{\prec' F}$. Hence, the NRC abstractions are sound for verification under specified failures.

Hierarchy of NRC abstractions. The least precise NRC abstraction (using $\prec^*$) does not model the route selection procedure at all, and chooses any route. More precise abstractions can be obtained by modeling the route selection procedure *partially*. Figure 4 shows partial orders and corresponding route selection procedures (shown as steps in a ranking function) for OSPF and BGP, ordered from least precise ($\prec^*$) to most precise ($\prec$). For example, $\prec_{(lp,pl)}$ corresponds to the first two steps of BGP's route selection procedure, *i.e.,* it first finds routes with maximum local preference, and from these, selects one with minimum path length. Appendix A has more details of BGP's route selection procedure. Abstractions higher up in the hierarchy are more precise as they model more of the route selection procedure but are more expensive as their SMT encodings have more variables and constraints. This tradeoff between precision and performance is evident in our experiments: verification with $\prec_{(lp)}$ was successful for all networks for which verification with $\prec^*$ gave false positives (§VI-B), but took up to 2.7x more time.

Abstraction refinement. If verification with an abstraction fails, we *validate* the returned counterexample by checking

if each node actually chose the best route. Note that the selected routes in the counterexample may contain only some fields, depending on the abstraction used. We first find the values of the other fields and the set of available routes by applying the transfer functions along the edges in the counterexample, starting from the destination router (*i.e.,* by effectively simulating the counterexample on the concrete SRP). We then check if all routers selected the best route that they received. If this is the case, we have found a *real* counterexample, *i.e.,* a stable solution in the real network that violates the property; if not, the counterexample is *spurious*. We can eliminate the spurious counterexample by adding a blocking clause that is the negation of the variable assignment corresponding to it and repeat verification with the same abstraction in a CEGAR [35] loop, but this could take many iterations to terminate. Instead, we suggest choosing a more precise abstraction which is higher up in the NRC hierarchy. We could potentially use a *local* refinement procedure that uses a higher-precision abstraction only at certain routers, based on the counterexample. We plan to explore this and other ways of counterexample-guided abstraction refinement in future work.

V. SMT Encodings

In this section we present our SMT encodings for an abstract SRP based on symbolic graphs. SRPs [30] can model many distributed routing protocols (*e.g.,* RIP, BGP, etc.) where the protocol and configurations determine the partial order for route selection and the transfer function. We begin by providing definitions for a symbolic graph and its solutions.

Definition 3 (Symbolic graph [21])**.** A symbolic graph $\mathcal{G}_{RE}$ is a tuple (G, RE) where $G = (V, E)$ is a graph and $RE = \{re_{uv} | (u, v) \in E\}$ is a set of Boolean routing edge variables.

Definition 4 (Symbolic graph solutions [21])**.** A symbolic graph $\mathcal{G}_{RE} = (G, RE)$ and a formula F over RE has solutions $Sol(\mathcal{G}_{RE},\ F)$ which are subgraphs of G defined by assignments to RE that satisfy F, such that an edge (u, v) is in a solution subgraph iff $re_{uv} = 1$ in the corresponding satisfying assignment.

A. Routing Constraints on Symbolic Graphs

We now describe the constraints in our SMT formulation, $\widehat{N}$, of the abstract SRP $\widehat{S}$. The symbolic graph solutions $Sol(G_{RE}, \widehat{N})$ correspond to solutions of $\widehat{S}$. The complete formulation is summarized in Figure 5.

- **Routing choice constraints:** Each node other than the destination chooses a neighbor to route through or $None$, which denotes no route (eqn. 2). We use a variable $nChoice$ to denote a node's choice. The routing edge re_{vu} is true iff u chooses a route from v (eqn. 3).
- **Route availability constraints:** If a node u chooses to route through a neighbor v, then v must have a route to the destination (eqn. 5). If every neighbor v either has no route ($\neg hasRoute_v$) or the route is dropped ($routeDropped_{vu}$), then u must choose $None$ (eqn. 6).
- **Attribute transfer and route filtering constraints:** If u chooses to route through neighbor v (*i.e.,* $re_{vu} = 1$), the transfer function relates their attributes and v's route must not be dropped along edge (v, u) (eqns. 7 and 8). The attribute at the destination is the initial route a_d (eqn. 9).

Our formulation is parameterized by three placeholders: (1) $hasRoute_v$, which is true iff v receives a route from the destination; (2) $trans_{vu}$, the transfer function along edge (v, u); and (3) $routeDropped_{vu}$, which is true iff the route is filtered along the edge (v, u). Of these, $trans_{vu}$ and $routeDropped_{vu}$ depend on the network protocol and configuration, and are shown in an example below. The encodings of $hasRoute$ are described in the next subsection.

Example 5 (Transfer constraints)**.** The attribute transfer and route filtering constraints in the abstract SRP (with partial order $\prec^*$) are shown below for the network in Figure 1b.

We only model fields used in route filtering (*i.e.,* the community attribute) and ignore local preference and path length. We use a bit vector variable $comm_u$ to denote the community attribute at node R_u, and a Boolean routing edge variable re_{uv} for each edge (R_u, R_v). We encode the presence of community tag $c1$ as 1, and its absence as 0.

Initial route at destination. We set the community attribute to 0 at the destination R_1 using the constraint $comm_1 = 0$.

Transfer constraints along edge (R_1, R_3)**.** The transfer function adds the community tag $c1$. The route is never dropped along this edge, so the placeholder $routeDropped_{13}$ is false.

$$re_{13} \rightarrow comm_3 = 1 \tag{15}$$
$$re_{13} \rightarrow \neg routeDropped_{13} \tag{16}$$
$$routeDropped_{13} \leftrightarrow False \tag{17}$$

Our implementation simplifies formulas when $routeDropped$ is a constant, and only asserts equation (15) above.

Transfer constraints along edges (R_5, R_7) **and** (R_6, R_7)**.** The transfer functions propagate the community attribute and filter routes based on whether tag $c1$ is present.

$$re_{57} \rightarrow comm_7 = comm_5 \tag{18}$$
$$re_{57} \rightarrow \neg routeDropped_{57} \tag{19}$$
$$routeDropped_{57} \leftrightarrow (comm_5 = 1) \tag{20}$$
$$re_{67} \rightarrow comm_7 = comm_6 \tag{21}$$
$$re_{67} \rightarrow \neg routeDropped_{67} \tag{22}$$
$$routeDropped_{67} \leftrightarrow (comm_6 = 0) \tag{23}$$

Transfer constraints along other edges. The transfer functions propagate the community attribute and do not filter routes.

$$re_{vu} \rightarrow comm_u = comm_v \tag{24}$$
$$re_{vu} \rightarrow \neg routeDropped_{vu} \tag{25}$$
$$routeDropped_{vu} \leftrightarrow False \tag{26}$$

Our implementation simplifies the formulas by substituting the value of $routeDropped$, and only asserts equation (24).

Abstract SRP $\widehat{S} = (G,\ A,\ a_d,\ \prec',\ trans),\ G = (V,\ E,\ d)$
Symbolic graph $\mathcal{G}_{RE} = (G,\ RE)$

Variables

$attr_u$: bit vector	*route announcement fields,* $\forall u \in V$
$nChoice_u$: bit vector	*neighbor choice,* $\forall u \in V \setminus \{d\}$
$hasRoute_u$: Boolean	*placeholder for route availability,* $\forall u \in V$
$routeDropped_{uv}$: Boolean	*route dropped along an edge,* $\forall (u,v) \in E$

Constants

$nID(u,v)$: integer	*u's neighbor ID for v,* $\forall (u,v) \in E$
$None_u$: integer	*ID denoting no neighbor,* $\forall u \in V$

Routing choice constraints

$$\left(\bigvee_{(v,u)\in E} nChoice_u = nID(u,v)\right) \vee nChoice_u = None_u \tag{2}$$

$$nChoice_u = nID(u,v) \leftrightarrow re_{vu} \tag{3}$$

$$\neg re_{vd} \quad \forall (v,d) \in E \tag{4}$$

Route availability constraints

$$nChoice_u = nID(u,v) \rightarrow hasRoute_v \tag{5}$$

$$nChoice_u = None_u \leftrightarrow \bigwedge_{(v,u)\in E} \neg hasRoute_v \vee routeDropped_{vu} \tag{6}$$

Attribute transfer and route filtering constraints

$$re_{vu} \rightarrow attr_u = trans_{vu}(attr_v) \tag{7}$$

$$re_{vu} \rightarrow \neg routeDropped_{vu} \tag{8}$$

$$attr_d = a_d \tag{9}$$

Solver-specific constraints

(a) SMT solvers with graph theory support (*e.g.,* MonoSAT):

$$\forall u \in V,\ hasRoute_u \leftrightarrow \mathcal{G}_{RE}.reaches(d,u) \tag{10}$$

(b) SMT solvers without graph theory support (*e.g.,* Z3):

$$hasRoute_d \tag{11}$$

$$\forall u \neq d,\ hasRoute_u \leftrightarrow \bigvee_{v,\ (v,u)\in E} hasRoute_v \wedge re_{vu} \tag{12}$$

$$rank_d = 0 \tag{13}$$

$$\forall (v,u) \in E,\ re_{vu} \rightarrow rank_u = (rank_v + 1) \tag{14}$$

Fig. 5: Symbolic graph-based encoding for an abstract SRP.

B. Solver-specific Constraints

We have two encodings of $hasRoute$, depending on whether the SMT solver has graph theory support.

SMT solvers with graph theory support. We use the reachability predicate $\mathcal{G}_{RE}.reaches$ to encode $hasRoute$: $hasRoute_v$ is true iff $\mathcal{G}_{RE}.reaches(d,v)$ (*i.e.,* there is a path from d to v in the symbolic graph $\mathcal{G}_{RE}$), where d is the destination (eqn. 10). Additionally, we use the reachability predicate to model regular expressions over paths, which most tools do not support. For example, the regular expression ".*ab.*c.*d.*" (where '.' matches any character and '*' denotes 0 or more occurrences of the preceding character) matches any path that traverses edge (a, b), node c, and then node d, and is encoded as $re_{ab} \wedge \mathcal{G}_{RE}.reaches(b,c) \wedge \mathcal{G}_{RE}.reaches(c,d)$.

Standard SMT solvers. We interpret $hasRoute$ as a reachability marker which indicates whether a route has been received and add constraints to propagate the marker in the symbolic graph (eqns. 11 and 12). To prevent solutions with loops, we use a variable, $rank$, at each node to track the path length along with additional constraints (eqns. 13 and 14).

Loop prevention. In BGP, routing loops are prevented using the AS path attribute, the list of autonomous systems (ASes) in the route; routers drop routes if the AS path contains their AS. To model BGP's loop prevention mechanism exactly, Minesweeper's [11] SMT encoding would require $O(N^2)$ additional variables (where N is the number of routers) to track for each router, the set of routers in the AS path. Since this is expensive, Minesweeper uses an optimization that relies on the route selection procedure to prevent loops when routers use default local preference: the shorter loop-free path will be selected. Our encodings for $hasRoute$ model BGP's loop prevention mechanism exactly with fewer additional variables: the MonoSAT encoding uses no additional variables and the Z3 encoding uses $O(N)$ additional variables ($rank$).

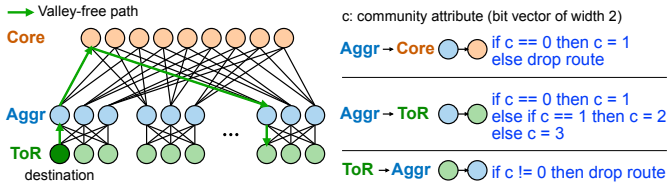
C. Benefits of the NRC Abstractions in SMT Solving

Fewer attributes. The most direct benefit is that with NRC abstractions many route announcement fields become irrelevant and can be removed from the network model, resulting in smaller SMT formulas. Specifically, all fields required to model *route filtering* (*i.e.,* the dropping of route announcements) and the property of interest are retained, but fields used only for route selection (*e.g.,* local preference) can be removed depending on the specific abstraction.

Expensive transfers can be avoided during SMT search. Once a neighbor is selected during the SMT search, then transfers of attributes from other neighbors become irrelevant. In contrast, without any abstraction, each node *must* consider transfers of attributes from *all* neighbors to pick the best route.

D. Encoding Properties for Verification

Reachability. We encode the property that a node u can reach destination d by asserting its negation: $nChoice_u = None_u$.



(a) FatTree topology (b) Valley-free policy

Fig. 6: Example data center network with a valley-free policy.

Non-reachability/Isolation. We encode the property that a node u can never reach d by asserting $nChoice_u \neq None_u$.

No-transit property. Routing policies between autonomous systems (ASes) are typically influenced by business relationships such as provider-customer or peer-peer [19], [36]. A provider AS is paid to carry traffic to and from its customers while peer ASes exchange traffic between themselves and their customers without any charge. The BGP policies (Gao-Rexford conditions [19]) between ASes usually ensure that an AS does not carry traffic from one peer or provider to another. This is called a *no-transit* property; its negation is encoded as $\bigvee_{u \in V} \bigvee_{\substack{v,w \in PeerProv(u), \\ v \neq w}} re_{vu} \wedge re_{uw}$, where $PeerProv(u)$ denotes neighbors of u that are its peers or providers.

Policy properties. BGP policies can be defined by assigning meaning to specific community tags. Policy properties can then be encoded using formulas over the communities at a node.

Example 6 (Valley-free Policy)**.** The valley-free policy prevents paths that have valleys, *i.e.,* paths which go up, down, and up again between the layers of a FatTree network topology [23], [29]; a valley-free path is shown in Figure 6a. Figure 6b shows an implementation of the valley-free policy where c denotes the community attribute in BGP. A path between ToR routers with a valley between the Aggr and Core layers will cross an Aggr router at least thrice, updating c to 3. Hence, the negation of the valley-free property at a node u is encoded as $comm_u = 3$.

VI. Implementation and Evaluation

We implemented our abstractions and SMT encodings in a prototype tool called ACORN, with backends to MonoSAT and Z3 solvers. (The SMT encoding for an abstract SRP (§V) is extended for a concrete SRP using additional constraints described in Appendix D.) ACORN's input is an intermediate representation (IR) of a network topology and configurations (described in Appendix C) which represents routing policy using match-action rules, similar to route-maps in Cisco's configuration language, and could serve as a target for frontends such as Batfish [14] or NV [13] in the future.

In our evaluation, we measure the effectiveness of the NRC abstractions and use two back-end SMT solvers – MonoSAT and Z3 (with bitvector theory and bit-blasting enabled). We use four settings: (1) **abs_mono**: with NRC abstraction ($\prec^*$), using MonoSAT; (2) **abs_z3**: with NRC abstraction ($\prec^*$), using Z3; (3) **mono**: without abstraction, using MonoSAT; (4) **z3**: without abstraction, using Z3. We evaluated ACORN on two types of benchmarks: (1) data center networks with FatTree topologies [23] (a commonly used topology), and (2) wide area networks from Topology Zoo [24] and BGPStream [25] (more details are in Appendix C). We also compared ACORN with two state-of-the-art control plane verifiers on the data center benchmarks. All experiments were run on a Mac laptop with a 2.3 GHz Intel i7 processor and 16 GB memory.

A. Data Center Networks

We generated data center network benchmarks with FatTree topologies [23], with 125 to 36,980 nodes running four policies: (1) shortest-path routing policy, (2) valley-free policy, (3) an extension of the valley-free policy with an isolation property – it uses regular expressions to enforce isolation between a FatTree pod and an external router connected to the core routers, and (4) a buggy valley-free policy in which routers in the last pod cannot reach routers in other pods. We checked reachability for all policies, and a policy-based property for (2) and (3). The results are shown in Figure 7, with each graph showing the number of nodes on the x-axis and the verification time (in seconds) on the y-axis.

Our results show that for *all* data center examples, and with *both* solvers, *using the NRC abstraction is uniformly better than using the no-abstraction setting*. In particular, with the MonoSAT solver, the NRC abstraction can achieve a relative speed-up of 52x for verifying reachability (when verification completes within a 1 hour timeout). Note also that MonoSAT performed better than Z3 by up to 10x; leveraging graph-based reasoning was clearly beneficial for these examples. Our abstract settings successfully verified all properties without any false positives, showing that the NRC abstraction can handle realistic policies. For networks running the buggy valley-free policy, our tool correctly reports that the destination is unreachable (results are in Figure 7f). Furthermore, our abstraction is effective even in these cases: abs_mono finishes on 3,000 nodes within an hour, while both no-abstraction settings time out on 2,000 nodes.

In terms of scalability, for both solvers, the no-abstraction setting times out beyond 4,500 nodes for reachability verification, while the abstract setting scales up to about 37,000 nodes for the shortest-path and valley-free policies, and up to 18,000 nodes for the isolation policy. To the best of our knowledge, all prior related work has been shown on networks with up to 4,500 nodes (maximum), which are much smaller than large data centers in operation today.

B. Wide Area Networks

To evaluate ACORN on less regular network topologies than data centers, we considered wide area network benchmarks. These typically have small sizes and are not easily parameterized, unlike data center topologies. We evaluated ACORN on two sets of wide area networks: (1) 10 of the larger networks from Topology Zoo [24], with 22 to 79 routers, which we annotated with business relationships (since Topology Zoo only provides topologies), and (2) 10 example networks based

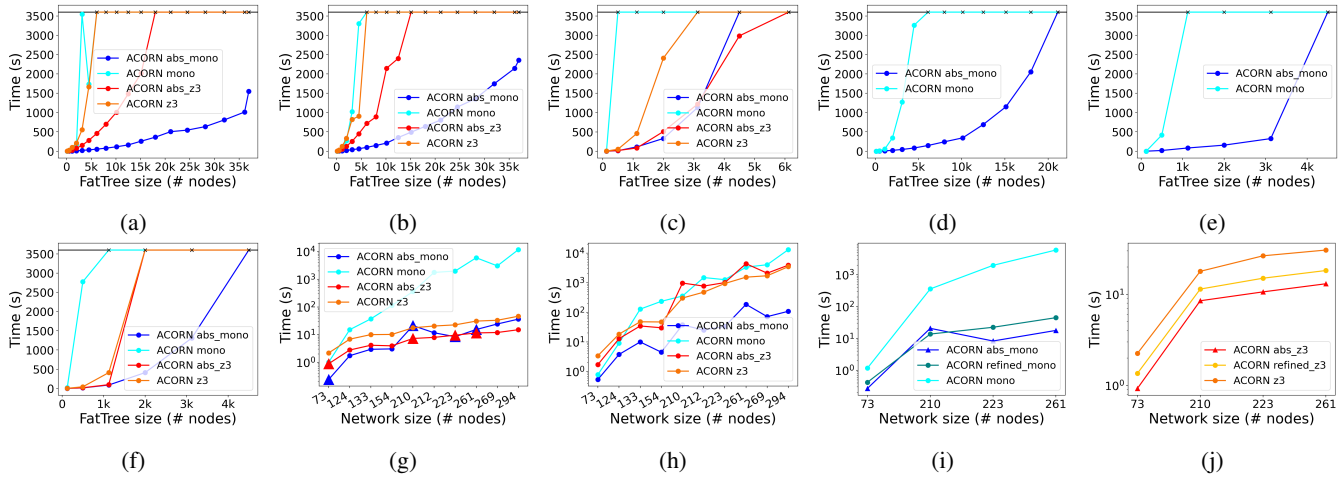


Fig. 7: Results for data center networks: (a) reachability with shortest-path routing, (b) reachability with valley-free policy, (c) valley-free property, (d) reachability with isolation policy, (e) isolation property, and (f) reachability with a buggy valley-free policy. Results for BGPStream examples: (g) reachability, (h) no-transit property, refinement using (i) MonoSAT and (j) Z3.

on parts of the Internet that were involved in misconfiguration incidents as reported on BGPStream [25], which we annotated with publicly available business relationships (CAIDA AS relationships dataset [37]). For all benchmarks, we used a BGP policy that implements the Gao-Rexford conditions [19]: (1) routes from peers and providers are not exported to other peers and providers, and (2) routes from customers are preferred over routes from peers, which are preferred over routes from providers. We then checked two properties: reachability of all nodes to a destination, and the no-transit property (§V-D).

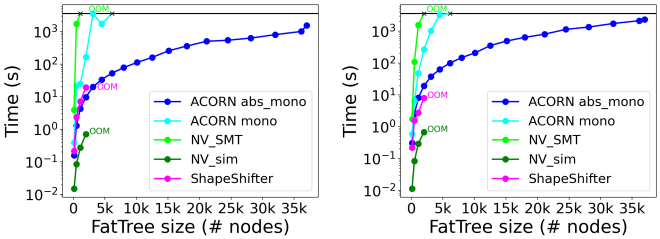
Topology Zoo benchmarks. The abstract settings successfully verify both properties and are up to 3x faster than the respective no-abstraction settings. All settings take less than 0.5s for both properties (detailed results are in Appendix C).

BGPStream benchmarks. The results are in Figures 7g to 7j, with the number of nodes (ASes) on the x-axis and verification time in seconds on the y-axis (log scale). The abstract settings successfully verified reachability in 6 networks and gave false positives (denoted by triangular markers) for 4; when successful, the abstract settings performed much better than the no-abstraction settings with relative speedups of up to 323x for MonoSAT and 3x for Z3. For the no-transit property, abs_mono is up to 120x faster than mono, while abs_z3 is faster than z3 for some networks but slower for others.

For the 4 benchmarks with false positives, we used a more precise abstraction, $\prec_{(lp)}$, which models local preference (results shown in Figures 7i and 7j). Our $\prec_{(lp)}$ abstraction is successful on all 4 networks, with relative speedups (over no abstraction) of up to 133x for MonoSAT and 1.8x for Z3, and relative slowdowns (over $\prec^*$) of up to 2.7x for MonoSAT and 1.5x for Z3. These results demonstrate the precision-cost tradeoff enabled by the NRC abstraction hierarchy.

C. Comparison with Existing Tools

We compared ACORN with two state-of-the-art control plane verifiers: ShapeShifter [16] and NV [13] (FastPlane [15] and Hoyan [18] are not publicly available). ShapeShifter uses simulation with abstract interpretation [38], with binary decision diagrams (BDDs) [39] representing sets of abstract routing messages. NV is a functional programming language for modeling and verifying network control planes. It provides a simulator (based on Multi-Terminal BDDs [40] but without abstraction of routing messages) and an SMT-based verifier that uses Z3. (NV's SMT engine has been shown to perform better than Minesweeper [13].) NV uses a series of front-end transformations to generate an SMT formula (we only report NV's SMT solving time), but its encoding is not based on symbolic graphs. A comparison of our no-abstraction settings against NV_SMT gives some indication of the effectiveness of our SMT encoding. We performed experiments on the data center benchmarks (§VI-A), where we generated corresponding inputs for ShapeShifter and NV with the same routing message fields. The results for the shortest-path routing and valley-free policies are shown in Figure 8, with the number of nodes shown on the x-axis, verification time in seconds on the y-axis (log scale), timeouts denoted by 'x', and out-of-memory denoted by 'OOM'. (ShapeShifter and NV could not be run on the isolation benchmarks as they do not support regular expressions over AS paths.) Note that both NV and ShapeShifter run out of memory for networks with more than 3,000 nodes while ACORN's mono and abs_mono settings can



(a) Shortest-path policy (b) Valley-free policy

Fig. 8: Comparison of tools on data center examples.

verify larger networks with 4,500 nodes and 36,980 nodes, respectively. These results show that SMT-based methods for network control plane verification can scale to large networks with tens of thousands of nodes.

D. Discussion and Limitations

ACORN is sound for properties that hold for all *stable* states of a network, *i.e.,* properties of the form $\forall s\ P(s)$ where s is a stable state, such as reachability, policy-based properties, device equivalence, and way-pointing. Like many SMT-based tools, ACORN cannot verify properties over *transient* states that arise before convergence. For checking reachability, our least precise abstraction works well in practice; to verify a property about the path length between two routers a user should use an abstraction that models path length (otherwise our verification procedure would give a false positive). We have shown that our abstractions are sound under specified failures; however, our tool does not yet model failures, which we plan to consider in future work.

VII. Related Work

Our work is related to other efforts in network verification and the use of nondeterministic abstractions for verification.

Distributed control plane verification. These methods [41], [12], [42], [17], [11], [13] aim to verify all data planes that emerge from the control plane. Simulation-based tools [14], [43], [15] can scale to large networks, but can miss errors that are triggered only under certain environments. The FastPlane [15] simulator scales to large data centers (results shown for ≈2000 nodes) but it requires the network policy to be monotonic [31] (a route announcement's preference decreases along any edge in the network) while our approach does not. Hoyan [18] uses a hybrid simulation and SMT-based approach which tracks multiple routes received at each router to check reachability under failures, but in the context of the given simulation. The ShapeShifter [16] work is the closest to ours in terms of route abstractions, but it does not scale as well as our tool (§VI-C). Moreover, our SMT-based approach provides better precision by exploring multiple routing choices at each node and tracking correlations across different nodes, whereas ShapeShifter uses a conservative abstraction at each node, much as SMT-based program verification allows path-sensitivity for more precision than path-insensitive static analysis. For example, ShapeShifter's ternary abstraction (which abstracts each community tag bit to $\{0, 1, *\}$) would result in a false positive on Example 2 (§II), while ACORN verifies it correctly. Bagpipe [12] verifies BGP policies using symbolic execution and uses a simplified BGP route selection procedure that chooses routes with maximum local preference, similar to our NRC abstraction using $\prec_{(lp)}$. Our abstraction hierarchy is more general and can be applied to any routing protocol.

ARC [44] and QARC [45] use a graph-based abstraction combined with graph algorithms and mixed-integer linear programming respectively, but do not support protocol features such as local preference and community tags. Tiramisu [46] uses a similar graph-based representation, but with multiple layers to capture inter-protocol dependencies and was shown to scale to networks with a few hundred devices. Bonsai [30] compresses the network control plane to take advantage of symmetry in the network topology and policy; NRC abstractions can be applied even when the network is not symmetric.

Some recent approaches [47], [48], [20] use modular verification techniques to improve the scalability of verification; the core ideas in modular verification are orthogonal to our work. Among these efforts, Lightyear [20] also verifies BGP policies using an over-approximation that allows routers to choose any received route – this corresponds to our NRC abstraction with partial order $\prec^*$. However, unlike our approach, it requires a user to provide suitable invariants.

Data plane verification. These efforts [49], [50], [51], [52], [53], [54], [55], [56] model the data forwarding rules and check properties such as reachability, absence of routing loops, etc. Many such methods have been shown to successfully handle the scale and complexity of real-world networks. Similar to these methods, our least precise abstraction does not model the route selection procedure but we verify all data planes that emerge from the control plane, not just one snapshot.

Nondeterminism and abstractions. Nondeterministic abstractions have been used in many different settings in software and hardware verification. Examples include control flow nondeterminism in Boolean program abstractions in SLAM [57], a sequentialization technique [58] that converts control nondeterminism (*i.e.,* interleavings in a concurrent program) to data nondeterminism, and a localization abstraction [59] in hardware designs. Our NRC abstractions use route nondeterminism to soundly abstract network control plane behavior.

VIII. Conclusions and Future Directions

The main motivation for our work is to improve the scalability of symbolic verification of network control planes. Our approach is centered around two core contributions: a hierarchy of nondeterministic routing choice abstractions, and a new SMT encoding that can leverage specialized SMT solvers with graph theory support. Our tool, ACORN, has verified reachability (an important property for network operators) on data center benchmarks (with FatTree topologies and commonly used policies) with ≈37,000 routers, which far exceeds what has been shown by existing related tools. Our evaluation shows that our abstraction performs *uniformly better* than no abstraction for verifying reachability for different network topologies and policies, and with two different SMT solvers. In future work, we plan to consider verification under failures, and combine our abstractions with techniques based on modular verification of network control planes.

Acknowledgments

This work was supported in part by NSF Grants 1837030 and 2107138. Any opinions, findings, and conclusions expressed herein are those of the authors and do not necessarily reflect those of the NSF. We would also like to thank Anish Athalye for permitting use of Basalt, a language for graphic design.

APPENDIX A
BGP OVERVIEW

BGP is the protocol used for routing between *autonomous systems* (ASes) in the Internet. An autonomous system (AS) is a network controlled by a single administrative entity, *e.g.,* the network of an Internet Service Provider (ISP) in a particular country, or a college campus network. A simplified version of the decision process used to select best routes in BGP is shown in Table I [36]. A router compares two route announcements by comparing the attributes in each row of the table, starting from the first row. A route announcement with higher local preference is preferred, regardless of the values of other attributes; if two route announcements have equal local preference, then their path lengths will be compared. BGP allows routes to be associated with additional state via the community attribute, a list of string tags. Decisions can be taken based on the tags present in a route announcement; for example, a route announcement containing a particular tag can be dropped or the route preference can be altered (e.g., by increasing the local preference if a particular tag is present).

APPENDIX B
PROOF OF SOUNDNESS OF THE NRC ABSTRACTIONS

Lemma 1. [Over-approximation] For an SRP S and corresponding abstract SRP $\widehat{S}_{\prec'}$ with solutions $Sol(S)$ and $Sol(\widehat{S}_{\prec'})$ respectively, $Sol(S) \subseteq Sol(\widehat{S}_{\prec'})$.

Proof. We need to show that for each labeling $\mathcal{L}$, if $\mathcal{L} \in Sol(S)$ then $\mathcal{L} \in Sol(\widehat{S}_{\prec'})$. An SRP solution $\mathcal{L}$ is defined by

$$\mathcal{L}(u) = \begin{cases} a_d & \text{if } u = d \\ \infty & \text{if } \text{attrs}_{\mathcal{L}}(u) = \emptyset \\ a \in \text{attrs}_{\mathcal{L}}(u) \text{ , minimal by } \prec & \text{if } \text{attrs}_{\mathcal{L}}(u) \neq \emptyset \end{cases}$$

where $\text{attrs}_{\mathcal{L}}(u)$ is the set of attributes that u receives from its neighbors. The abstract SRP $\widehat{S}_{\prec'}$ differs from the SRP S only in the partial order. Therefore, to show that $\mathcal{L}$ is a solution of $\widehat{S}_{\prec'}$, we need to show that if $\text{attrs}_{\mathcal{L}}(u) \neq \emptyset$, then $\mathcal{L}(u)$ is minimal by $\prec'$. By the definition of an abstract SRP, the set of minimal attributes according to $\prec'$ is a superset of the set of minimal attributes according to $\prec$, which means $\mathcal{L}(u)$ is minimal by $\prec'$. Therefore, any SRP solution $\mathcal{L}$ is a solution of the abstract SRP $\widehat{S}_{\prec'}$. □

Theorem 1. [Soundness] Given SMT formulas $\widehat{N}$ and N modeling the abstract and concrete SRPs respectively and SMT formula P encoding the property to be verified, if $\widehat{N} \wedge \neg P$ is unsatisfiable, then $N \wedge \neg P$ is also unsatisfiable.

Proof. If $\widehat{N} \wedge \neg P$ is unsatisfiable, every solution of the abstract SRP satisfies the given property. By Lemma 1, the property also holds for all solutions of the concrete SRP S, *i.e.,* there is no property violation in the real network. □

APPENDIX C
ACORN INTERMEDIATE REPRESENTATION (IR) AND BENCHMARK EXAMPLES

Intermediate Representation (IR). Our IR represents a transfer function as a list of match-action rules, similar to route-maps in Cisco's configuration language. We support matching on the community attribute and some types of regular expressions over the AS path. Our implementation currently supports regular expressions that check whether the path contains certain ASes or a particular sequence of ASes, and could be extended to support general regular expressions in the future. A match can be associated with multiple actions, which can update route announcement fields such as the community attribute, local preference, and AS path length.

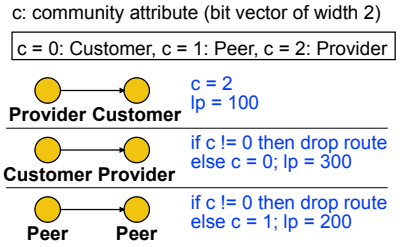
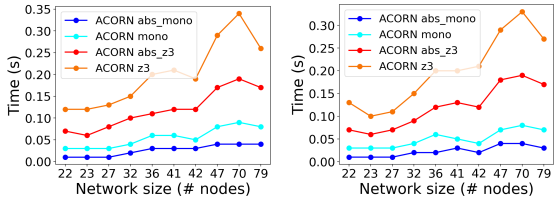


Fig. 9: BGP policy implementing Gao-Rexford conditions [19]



(a) Reachability (b) No-transit property

Fig. 10: Results for Topology Zoo examples.

Benchmark examples. The details of the wide area network examples we used (§VI-B) are described below.

Topology Zoo benchmarks. We used 10 topologies from the Topology Zoo [24], which we pre-processed, *e.g.,* by removing duplicate nodes and nodes with id "None". The details of the resulting topologies are shown in Table II. We annotated the topologies with business relationships, considering each node as an AS, and used a BGP policy that implements the Gao-Rexford conditions [19] (Figure 9). The annotated benchmark files (in GML format) are included in our benchmark repository, along with the examples in our IR format.

BGPStream benchmarks. We created a set of 10 examples based on parts of the Internet involved in BGP hijacking incidents, as reported on BGPStream [25]. For each hijacking incident, we created a network with the ASes involved and used the CAIDA AS Relationships dataset [37] to add edges between ASes with the given business relationships (customer-provider or peer-peer). We then removed some ASes (if required) so that our no-abstraction setting could verify that all ASes in the resulting network can reach the destination (taken to be the possibly hijacked AS). We used a BGP policy (shown in Figure 9) that implements the Gao-Rexford conditions [19]. The details of the examples are shown in Table III.

Results for Topology Zoo examples. Detailed results for the Topology Zoo benchmark examples are shown in Figure 10.

Step	Attribute	Description	Preference (Lower/Higher)
1	Local preference	An integer set locally and not propagated	Higher
2	AS path length	The number of ASes the route has passed through	Lower
3	Multi-exit Discriminator (MED)	An integer influencing which link should be used between two ASes	Lower
4	Router ID	Unique identifier for a router used for tie breaking	Lower

TABLE I: Simplified BGP decision process to select the best route [36].

Benchmark	Topology name	Size
TZ1	VinaREN	22 nodes, 24 edges
TZ2	FCCN	23 nodes, 25 edges
TZ3	GTS Hungary	27 nodes, 28 edges
TZ4	GTS Slovakia	32 nodes, 34 edges
TZ5	GRnet	36 nodes, 41 edges
TZ6	RoEduNet	41 nodes, 45 edges
TZ7	LITNET	42 nodes, 42 edges
TZ8	Bell South	47 nodes, 62 edges
TZ9	Tecove	70 nodes, 70 edges
TZ10	ULAKNET	79 nodes, 79 edges

TABLE II: Topology Zoo examples.

Benchmark	Incident date	Size
B1	2021-06-14	261 nodes, 3325 edges
B2	2021-06-17	223 nodes, 2722 edges
B3	2021-06-18	133 nodes, 1205 edges
B4	2021-06-19	210 nodes, 2100 edges
B5	2021-06-21	269 nodes, 3351 edges
B6	2021-06-22	212 nodes, 2233 edges
B7	2021-06-22	294 nodes, 4108 edges
B8	2021-06-22	124 nodes, 860 edges
B9	2021-06-22	73 nodes, 270 edges
B10	2021-06-25	154 nodes, 1176 edges

TABLE III: BGPStream examples.

APPENDIX D
SMT CONSTRAINTS FOR CONCRETE SRP

We extend our abstract SRP formulation (Figure 5) to encode a concrete SRP by adding additional constraints ensuring that each node picks the best route, *i.e.,* for every edge $(v, u) \in E$, if u selects the route from v then v's route must be the best route that u receives from its neighbors. This requires keeping track of the attribute fields used in route selection (such as path length) and possibly additional variables to track the minimum or maximum value of an attribute. The constraints required to model the first two steps in BGP's route selection procedure are shown in Example 7.

Example 7 (Encoding route selection in BGP)**.** We keep track of local preference (denoted lp) and AS path length (denoted $path$) and encode transfer constraints over these attributes (*e.g.,* to increment path length). For each edge (v, u), we use lp_{vu} to denote the local preference of the route sent from v to u after applying the transfer function. For each node u we use $maxLp_u$ to track the maximum local preference of routes node u receives, and $minPath_u$ to track the minimum path length among routes with the maximum local preference.

We define $maxLp_u$ below ($nValid_{vu} \leftrightarrow hasRoute_v \wedge \neg routeDropped_{vu}$ indicates whether v sends a route to u).

$$\bigwedge_{(v,u)\in E} nValid_{vu} \rightarrow maxLp_u \geq lp_{vu}$$

$$nChoice_u \neq None_u \rightarrow \bigvee_{(v,u)\in E} nValid_{vu} \wedge maxLp_u = lp_{vu}$$

We define $minPath_u$ using similar constraints:

$$\bigwedge_{(v,u)\in E} (nValid_{vu} \wedge lp_{vu} = maxLp_u) \rightarrow minPath_u \leq path_v$$

$$nChoice_u \neq None_u \rightarrow$$

$$\bigvee_{(v,u)\in E} nValid_{vu} \wedge lp_{vu} = maxLp_u \wedge minPath_u = path_v$$

We then add constraints to ensure that if u chooses a route from any neighbor v, then v's route must be the best.

$$nChoice_u = nID(u, v) \rightarrow lp_{vu} = maxLp_u \wedge path_v = minPath_u$$

REFERENCES

[1] N. Rockwell, "Summary of june 8 outage," https://www.fastly.com/blog/summary-of-june-8-outage, 2021.

[2] J. Graham-Cumming, "Cloudflare outage on july 17, 2020," https://blog.cloudflare.com/cloudflare-outage-on-july-17-2020/, 2020.

[3] M. Anderson, "Time warner cable says outages largely resolved," http://www.seattletimes.com/business/time-warner-cable-says-outages-largely-resolved, NY, NY, 2014.

[4] S. Ragan, "Bgp errors are to blame for monday's twitter outage, not ddos attacks," https://www.csoonline.com/article/3138934/security/bgp-errors-are-to-blame-for-monday-s-twitter-outage-not-ddos-attacks.html, 2016.

[5] D. Roberts, "It's been a week and customers are still mad at bb&t," https://www.charlotteobserver.com/news/business/banking/article202616124.html, 2018.

[6] Y. Sverdlik, "United says it outage resolved, dozen flights canceled monday," https://www.datacenterknowledge.com/archives/2017/01/23/united-says-it-outage-resolved-dozen-flights-canceled-monday, 2017.

[7] K. Jayaraman, N. Bjørner, J. Padhye, A. Agrawal, A. Bhargava, P.-A. C. Bissonnette, S. Foster, A. Helwer, M. Kasten, I. Lee, A. Namdhari, H. Niaz, A. Parkhi, H. Pinnamraju, A. Power, N. M. Raje, and P. Sharma, "Validating datacenters at scale," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: ACM, 2019, pp. 200–213.

[8] B. Tian, X. Zhang, E. Zhai, H. H. Liu, Q. Ye, C. Wang, X. Wu, Z. Ji, Y. Sang, M. Zhang, D. Yu, C. Tian, H. Zheng, and B. Y. Zhao, "Safely and automatically updating in-network acl configurations with intent language," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. Association for Computing Machinery, 2019, p. 214–226.

[9] H. Zeng, S. Zhang, F. Ye, V. Jeyakumar, M. Ju, J. Liu, N. McKeown, and A. Vahdat, "Libra: Divide and conquer to verify forwarding tables in huge networks," in *NSDI 14*, 2014.

[10] L. L. Peterson and B. S. Davie, *Computer Networks, Fifth Edition: A Systems Approach*, 5th ed., 2011.

[11] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *SIGCOMM*, Aug. 2017.

[12] K. Weitz, D. Woos, E. Torlak, M. D. Ernst, A. Krishnamurthy, and Z. Tatlock, "Formal semantics and automated verification for the border gateway protocol," in *NetPL*, 2016.

[13] N. Giannarakis, D. Loehr, R. Beckett, and D. Walker, "NV: An intermediate language for verification of network control planes," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2020, 2020, p. 958–973.

[14] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *NSDI*, 2015.

[15] N. P. Lopes and A. Rybalchenko, "Fast BGP simulation of large datacenters," in *Proc. of the 20th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, Jan. 2019.

[16] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "Abstract interpretation of distributed network control planes," *Proc. ACM Program. Lang.*, vol. 4, no. POPL, Dec. 2019.

[17] S. Prabhu, K. Y. Chou, A. Kheradmand, B. Godfrey, and M. Caesar, "Plankton: Scalable network configuration verification through model checking," in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2020, pp. 953–967.

[18] F. Ye, D. Yu, E. Zhai, H. H. Liu, B. Tian, Q. Ye, C. Wang, X. Wu, T. Guo, C. Jin, D. She, Q. Ma, B. Cheng, H. Xu, M. Zhang, Z. Wang, and R. Fonseca, "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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 599–614.

[19] L. Gao and J. Rexford, "Stable internet routing without global coordination," in *SIGMETRICS*, 2000.

[20] A. Tang, R. Beckett, K. Jayaraman, T. Millstein, and G. Varghese, "Lightyear: Using modularity to scale bgp control plane verification," *arXiv preprint arXiv:2204.09635*, 2022.

[21] S. Bayless, N. Bayless, H. H. Hoos, and A. J. Hu, "SAT modulo monotonic theories," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15, 2015, p. 3702–3709.

[22] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *TACAS*, 2008.

[23] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM*, 2008.

[24] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[25] "BGP Stream," https://bgpstream.com.

[26] "ACORN benchmark repository," https://github.com/divya-urs/ACORN_benchmarks.

[27] A. Abhashkumar, K. Subramanian, A. Andreyev, H. Kim, N. K. Salem, J. Yang, P. Lapukhov, A. Akella, and H. Zeng, "Running BGP in data centers at scale," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 2021, pp. 65–81.

[28] T. G. Griffin, F. B. Shepherd, and G. Wilfong, "The stable paths problem and interdomain routing," *IEEE/ACM Trans. Networking*, vol. 10, no. 2, 2002.

[29] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *SIGCOMM*, 2016.

[30] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "Control plane compression," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18, 2018, p. 476–489.

[31] J. a. L. Sobrinho, "An algebraic theory of dynamic network routing," *IEEE/ACM Trans. Netw.*, vol. 13, no. 5, pp. 1160–1173, Oct. 2005.

[32] T. G. Griffin and J. L. Sobrinho, "Metarouting," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2005, pp. 1–12.

[33] J. Backes, S. Bayless, B. Cook, C. Dodge, A. Gacek, A. J. Hu, T. Kahsai, B. Kocik, E. Kotelnikov, J. Kukovec, S. McLaughlin, J. Reed, N. Rungta, J. Sizemore, M. A. Stalzer, P. Srinivasan, P. Subotic, C. Varming, and B. Whaley, "Reachability analysis for aws-based networks," in *Computer Aided Verification (CAV), Proceedings, Part II*, 2019, pp. 231–241.

[34] S. Bayless, J. Backes, D. DaCosta, B. Jones, N. Launchbury, P. Trentin, K. Jewell, S. Joshi, M. Zeng, and N. Mathews, "Debugging network reachability with blocked paths," in *International Conference on Computer Aided Verification*. Springer, 2021, pp. 851–862.

[35] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *Computer Aided Verification, 12th International Conference, CAV, Proceedings*, 2000, pp. 154–169.

[36] M. Caesar and J. Rexford, "BGP routing policies in ISP networks," *Netwrk. Mag. of Global Internetwkg.*, vol. 19, no. 6, p. 5–11, Nov. 2005.

[37] "The CAIDA AS Relationships Dataset, May 1 2021," https://www.caida.org/catalog/datasets/as-relationships/.

[38] P. Cousot and R. Cousot, "Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, ser. POPL '77, 1977, p. 238–252.

[39] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[40] E. M. Clarke, M. Fujita, and X. Zhao, "Multi-terminal binary decision diagrams and hybrid decision diagrams," in *Representations of discrete functions*. Springer, 1996, pp. 93–108.

[41] A. Wang, L. Jia, W. Zhou, Y. Ren, B. T. Loo, J. Rexford, V. Nigam, A. Scedrov, and C. L. Talcott, "FSR: Formal analysis and implementation toolkit for safe inter-domain routing," *IEEE/ACM Trans. Networking*, vol. 20, no. 6, 2012.

[42] S. K. Fayaz, T. Sharma, A. Fogel, R. Mahajan, T. Millstein, V. Sekar, and G. Varghese, "Efficient network reachability analysis using a succinct control plane representation," in *OSDI*, 2016.

[43] B. Quoitin and S. Uhlig, "Modeling the routing of an autonomous system with c-bgp," *Netwrk. Mag. of Global Internetwkg.*, vol. 19, no. 6, pp. 12–19, Nov. 2005.

[44] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan, "Fast control plane analysis using an abstract representation," in *SIGCOMM*, 2016.

[45] K. Subramanian, A. Abhashkumar, L. D'Antoni, and A. Akella, "Detecting network load violations for distributed control planes," in *Proceedings of the ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI*, 2020, pp. 974–988.

[46] A. Abhashkumar, A. Gember-Jacobson, and A. Akella, "Tiramisu: Fast multilayer network verification," in *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, 2020, pp. 201–219.

[47] T. A. Thijm, R. Beckett, A. Gupta, and D. Walker, "Kirigami, the verifiable art of network cutting," *arXiv preprint arXiv:2202.06098*, 2022.

[48] ——, "Modular control plane verification via temporal invariants," *arXiv preprint arXiv:2204.10303*, 2022.

[49] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *NSDI*, 2012.

[50] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King, "Debugging the data plane with anteater," in *SIGCOMM*, 2011.

[51] E. Al-Shaer and S. Al-Haj, "FlowChecker: configuration analysis and verification of federated openflow infrastructures," in *3rd ACM Workshop on Assurable and Usable Security Configuration, SafeConfig 2010*, 2010, pp. 37–44.

[52] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *NSDI*, 2013.

[53] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte, "Real time network policy checking using header space analysis," in *NSDI*, Apr. 2013, pp. 99–112.

[54] C. J. Anderson, N. Foster, A. Guha, J.-B. Jeannin, D. Kozen, C. Schlesinger, and D. Walker, "NetKAT: Semantic foundations for networks," in *POPL*, 2014.

[55] S. Zhang and S. Malik, "SAT based verification of network data planes," in *Automated Technology for Verification and Analysis (ATVA)*, 2013.

[56] N. P. Lopes, N. Bjørner, P. Godefroid, K. Jayaraman, and G. Varghese, "Checking beliefs in dynamic networks," in *NSDI*, 2015.

[57] T. Ball, R. Majumdar, T. D. Millstein, and S. K. Rajamani, "Automatic predicate abstraction of C programs," in *Proceedings of the 2001 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2001, pp. 203–213.

[58] A. Lal and T. W. Reps, "Reducing concurrent analysis under a context bound to sequential analysis," in *Computer Aided Verification, 20th International Conference, CAV, Proceedings*, 2008, pp. 37–51.

[59] E. M. Clarke, R. P. Kurshan, and H. Veith, "The localization reduction and counterexample-guided abstraction refinement," in *Time for Verification, Essays in Memory of Amir Pnueli*, 2010, pp. 61–71.