Abstract Interpretation of Distributed Network Control Planes

RYAN BECKETT, Microsoft Research, USA
AARTI GUPTA, Princeton University, USA
RATUL MAHAJAN, University of Washington, USA and Intentionet, USA
DAVID WALKER, Princeton University, USA

The control plane of most computer networks runs distributed routing protocols that determine if and how traffic is forwarded. Errors in the configuration of network control planes frequently knock down critical online services, leading to economic damage for service providers and significant hardship for users. Validation via ahead-of-time simulation can help find configuration errors but such techniques are expensive or even intractable for large industrial networks. We explore the use of abstract interpretation to address this fundamental scaling challenge and find that the right abstractions can reduce the asymptotic complexity of network simulation. Based on this observation, we build a tool called ShapeShifter for reachability analysis. On a suite of 127 production networks from a large cloud provider, ShapeShifter provides an asymptotic improvement in runtime and memory over the state-of-the-art simulator. These gains come with a minimal loss in precision. Our abstract analysis accurately predicts reachability for *all* destinations for 95% of the networks and for most destinations for the remaining 5%. We also find that abstract interpretation of network control planes not only speeds up existing analyses but also facilitates new kinds of analyses. We illustrate this advantage through a new destination "hijacking" analysis for the border gateway protocol (BGP), the globally-deployed routing protocol.

CCS Concepts: • **Networks** → **Protocol testing and verification**; • **Software and its engineering** → **Abstraction, modeling and modularity**; **Software verification**; **Automated static analysis**.

Additional Key Words and Phrases: Network Verification, Network Simulation, Network Reliability, Abstract Interpretation, Network Control Plane, Distributed Routing Protocols, Router Configuration Verification

ACM Reference Format:
Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2020. Abstract Interpretation of Distributed Network Control Planes. *Proc. ACM Program. Lang.* 4, POPL, Article 42 (January 2020), 27 pages. https://doi.org/10.1145/3371110

1 INTRODUCTION

Computer networks are the connective tissue that provides access to services from online banking to retail to flight reservations to movies to local and federal governments. When they malfunction, these services often become inaccessible. The real-world consequences can range from revenue losses at cloud providers reaching $100K/minute [Tweney 2013] to mass groundings of flights [Sverdlik 2017] to customers being unable to withdraw funds from their bank accounts [Roberts 2018].

Authors' addresses: Ryan Beckett, Microsoft Research, USA, rbeckett@princeton.edu; Aarti Gupta, Princeton University, USA, aartig@princeton.edu; Ratul Mahajan, University of Washington, USA , Intentionet, USA, ratul@cs.washington.edu; David Walker, Princeton University, USA, dpw@princeton.edu.

2475-1421/2020/1-ART42
https://doi.org/10.1145/3371110

A fundamental hurdle to operating computer networks correctly is that they usually run distributed protocols, and the behavior of these protocols is controlled via hundreds of thousands, or sometimes even millions, of lines of low-level router configuration code.[1] At this scale, it is impossible to manually analyze configurations and certify their correctness. Moreover, any one-time manual analysis is woefully insufficient because the network is in constant flux—link and router failures change the paths used; new route announcements from a neighboring network change traffic patterns; and network operators change configurations to defend against security threats, engage in maintenance activities, reduce costs, or optimize performance. Such changes can lead to unexpected results and network outages [Sharwood 2016; Sverdlik 2012].

Naturally then, researchers have turned to automated analysis methods to address this network reliability challenge. In a first wave of research, researchers developed methods to analyze the network *data plane*, i.e., the set of rules that govern how network traffic is forwarded from point *A* to *B right now*. Systems such as Anteater [Mai et al. 2011], Header Space Analysis (HSA) [Kazemian et al. 2012], Veriflow [Khurshid et al. 2013], NetPlumber [Kazemian et al. 2013], NetKAT [Anderson et al. 2014], Network Optimized Datalog (NoD) [Lopes et al. 2015] and others model the data plane and check properties such as reachability, absence of black holes, equivalence and more. This groundbreaking research has produced methods that can scale to the world's largest networks.

However, networks have a meta component, the *control plane*, composed of distributed routing protocols that collectively decide which paths to use at any given time. When a failure happens, for instance, the control plane computes alternate paths that avoid the failure and installs them in the data plane. Hence, even if the data plane was correct 10 seconds ago, it may not be correct now. Indeed, in practice, faults in control planes can lie dormant only to be triggered when devices fail or during unexpected interactions with peer networks. A *control plane analysis* examines the configurations of the distributed routing protocols and attempts to verify that some, or all, of the possible future data planes that may be produced by a control plane satisfy a given property.

Research on control plane analysis is also well under way [Beckett et al. 2017, 2018; Fayaz et al. 2016; Feamster 2005; Fogel et al. 2015; Gember-Jacobson et al. 2016; Narain et al. 2016; Prabhu et al. 2017; Wang et al. 2012; Weitz et al. 2016]. These analyses typically considers one or more network environments (*i.e.*, failure scenarios and routing announcements from neighbors), predict the network behavior, and check that it satisfies desired connectivity and other invariants. They have been able to find a range of configuration errors in real networks.

Scalability remains a key challenge for control plane analysis, however. The fundamental issue is that the time and memory it takes to compute the behavior of routing protocols can grow highly non-linearly in the size of the network, even for fixed concrete environments [Fabrikant et al. 2011; L. Daggitt and Griffin 2018], let alone for general verification that considers an exhaustive set of environments. This observation is not merely theoretical. Tools that model the control plane precisely such as Minesweeper [Beckett et al. 2017] and Bagpipe [Weitz et al. 2016], exhibit highly non-linear performance curves and are thus unable to analyze large networks. Other tools, such as ARC [Gember-Jacobson et al. 2016] and ERA [Fayaz et al. 2016], scale better by sacrificing precision, *i.e.*, they craft specialized abstract representations that are either unsound or incomplete with respect to protocol features. Despite such approximations, their scaling behavior is still highly non-linear (*e.g.*, ERA must enumerate all pairs of network paths). Topological abstractions, in which an equivalent smaller network is crafted from a large network by exploiting symmetry [Beckett et al. 2018], can also help scale. But these lossless abstractions too bring limited relief and do not work when network topology or routing policy is asymmetric.

[1]Despite the emergence of software-defined networking (SDN), where the control plane is centralized, most large networks still use distributed protocols. We focus on such networks.

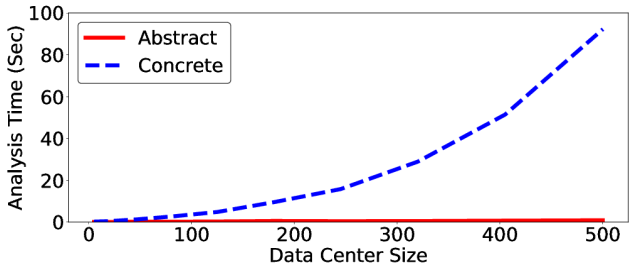

Fig. 1. Concrete vs. abstract analysis time.

Tackling the grand challenge of scalable control plane analysis requires systematically using abstraction to combat complexity. To do so, we build the first framework for analysis of network control planes via *abstract interpretation* [Cousot and Cousot 1977]. There are, of course, hundreds of papers on abstract interpretation of conventional programs. However, distributed control planes have a unique structure, semantics, and scaling properties, and they deserve special attention because they play a crucial role in the world's computing infrastructure. Our work does not contribute to the general literature on abstract interpretation but focuses on the use of abstract interpretation in the network control plane. It considers *which* abstractions to choose in this domain; *why* those abstractions are expected to work; *what* kinds of performance gains are expected; *when* these abstractions are sound; and *how* to implement a practical tool that exploits them.

Our most surprising and impactful result is that the right abstractions lead to asymptotic gains in performance for reachability analysis—a key property of interest to network operators—often with no loss of precision. Figure 1 plots the analysis time for our abstract simulator, ShapeShifter, against Batfish [Fogel et al. 2015], a state-of-the-art (concrete) network simulator, as the number of routers in the data center increases. Consistent with our complexity analysis, Batfish scales slightly worse than quadratically with network size whereas ShapeShifter scales slightly worse than linearly. Similar scaling trends hold for memory. The end result is that, in our experiments, Batfish runs out of memory on a machine with 32 GB of RAM when network sizes exceed 1000 devices, not an uncommon situation for a large network. On the other hand, ShapeShifter can continue to analyze these networks, and larger ones, in under a minute. We also found that our abstract reachability analysis was perfectly accurate on 95% of the 127 industrial networks we studied from a large cloud provider. On the remaining 5% of networks, we found the reachability analysis accurate for the majority of destinations (*i.e.*, IP prefixes).

Paper roadmap. In §2, we explain the basics of control plane protocols and our key insights, including *why* certain network abstractions can retain precision while speeding analysis and how abstract interpretation can enable new kinds of network analysis. In §3, we model network protocols as routing algebras [Griffin and Sobrinho 2005; Sobrinho 2005] and introduce the idea of abstract routing algebras. In §4, we list conditions relating concrete and abstract algebras that are sufficient for soundness. In §5, we enumerate, via examples, the space of control plane message abstractions that one may consider when crafting control plane analyses. In §6 and §7, we describe the design and implementation of ShapeShifter, the world's fastest control plane reachability analysis. Finally, in §8, we present results from running ShapeShifter on a range of synthetic and real networks.

2 KEY IDEAS

Networks with distributed control planes run routing protocols that compute paths to different destinations. While the protocols differ widely in the details, they share a common structure that can be formalized using *routing algebras* [Griffin and Sobrinho 2005; Sobrinho 2005]. In this section,

we first give an informal introduction to the process of modeling network control planes using routing algebras and then illustrate how to build abstractions atop these models. We then highlight three key observations that underpin our research: (1) despite only approximating control plane behavior, network abstractions often enable precise analysis for important properties, (2) they can improve analysis performance dramatically, and (3) they enable new kinds of analyses.

2.1 Routing Protocol Basics

Routing protocols such as BGP (Border Gateway Protocol), OSPF (Open Shortest Paths First), and RIP (Routing Information Protocol) are distributed systems in which nodes (routers) exchange *route announcements* for different destinations. Route announcements contain different types of information such as the path to the destination, the length of the path or an integer that encodes its relative desirability, and a tag to indicate the kind of the path (e.g., whether it flows through a customer, who will pay for traffic along the path, or alternately, a provider who will be paid). The goal is for each node to learn paths to various destinations and to select some of them for the purposes of forwarding data packets. For example, to forward a packet to a destination, a node will have to pick one of its neighbors as the next hop—routing protocols disseminate the information needed to make that choice. In this paper, we will use the word *message* or *route announcement* to describe the objects distributed by routing protocols and use the word *packet* to describe the application data (*e.g.*, the HTTP request) that is carried by the network.

Packets are routed to particular destination *IP addresses*. IP(v4) addresses are 32-bit numbers, written as four octets separated by periods, such as 10.10.10.1. In contrast, route announcements are associated with *IP prefixes*. An IP prefix is written as a pair of an IP address and length, such as 10.0.0.0/24. A prefix denotes a range of IP addresses that share the first L bits, where L is the length of the prefix. Hence, 10.0.0.0/24 indicates the first 24 bits are shared and includes addresses from 10.0.0.0 to 10.0.0.255. A route announcement for 10.0.0.0/24 tells a device where to send packets addressed to any of address in that range (which helps scale routing). In general, routers may have announcements for overlapping prefixes such as 10.0.0.0/24 and 10.0.0.0/16. For a given address, they select the longest-matching prefix. For instance, 10.0.0.0/24 will be selected over 10.0.0.0/16 for the address 10.0.0.1.

At a high level, all routing protocols work as follows. Each router sends messages to its neighbors about destinations it is directly connected to and about paths it will use for other destinations (if such a path exists). Upon receiving a set of messages containing routing information from its neighbors, each node selects the *best* message to use for each destination. Different protocols have different means of defining "best." RIP selects the path with the fewest hops; OSPF assigns cost to edges between nodes and chooses the least cost path; and BGP associates a number of attributes with each message, including local preference (a custom measure of desirability), a set of tags called communities, and a path, and determines "best" using a multi-step ranking function over these attributes. Before disseminating information about selected best messages to its neighbors, a node may modify the message on a per-neighbor basis. For instance, a BGP node could add a community to a message. Conversely, prior to using a received message for best path determination, the receiver may modify or filter it. For instance, a BGP node could change its local preference.

In well-designed networks, the process of propagating messages will eventually converge to a *stable* state where no node may improve the message it has currently selected—that is, none of its neighbors offer a message describing a superior route to the destination. The mapping from nodes to selected messages is called the *solution* to the control plane problem. Not all networks are guaranteed to converge or to converge to a unique solution. The convergence problem is well-studied and there are well-known criteria for determining convergence [Daggitt et al. 2018a].

While non-convergence can be problematic, it is not a major source of bugs in practice. We ignore issues of convergence in this paper and focus on other properties of interest.

Modeling routing protocols. To model a routing protocol (or multiple interacting protocols) on a network, we need to know: (1) the *topology* of the network, (2) the types of *messages* exchanged between nodes, (3) the *transfer function* (f), which describes how messages are transformed (or discarded) as they move between nodes, and (4) the *merge function* (written as $a \oplus b$), which describes how nodes combine messages to the same destination from multiple neighbors to select best messages.

With this information, one can simulate the execution of the routing protocol and find its *stable state*, by starting in the initial state, extracted from router configurations, and repeatedly exchanging messages between nodes. When node i sends a message to j, the transfer function f along the (i, j) edge will transform the message and the message will be merged with node j's current state. If j's current state is s and i's current state is m, when j receives a new message from i, its new state will be $f(m) \oplus s$. If $s = f(m) \oplus s$ then this part of the network is stable.

A simplified BGP example. Consider Figure 2(a), which presents the initial state for a 5-node BGP network. Each node is an independent autonomous system (AS), a configuration that is common in data centers [Lapukhov et al. 2015].[2] For simplicity, assume that the network has a single destination, R_1, and messages are represented as (`lp`, `path`, `comm`) triples, where `lp` is a local preference that BGP routers can set to prefer one route over another, `path` contains the AS path that routing advertisements traverse as they pass between devices, and `comm` is a bit vector that tracks the community values attached to the message. We also assume that there is only one possible community value, c_1, which is either present (1) or absent (0). R_1 starts by advertising the route `(100,[],[0])`, and all other nodes start with state ∞, meaning they have no route.

The default BGP transfer function along each edge (i, j) adds the AS number of the router (j) to the AS path and sets the local preference to 100. Network configurations can modify this default. In the figure, the network has been configured to add the community c_1 when a message passes between R_1 and R_3 and to update the local preference to 200 if the community c_1 is attached to messages passing between R_3 and R_4. The BGP merge function selects a route with the highest local preference, discarding messages with lower local preference. If two messages have the same local preference, it will choose the route with the shorter AS path.

Figure 2(b) records the stable state to which Figure 2(a) converges. The arrows indicate the flow of routing messages. More specifically, the (`lp`, `path`, `comm`)-triple printed in black next to each node is the final message chosen by that node (ignore the pairs printed in orange for now). For instance, router R_3 converges to `(100,[`R_1`],[1])`, a route with local preference (`lp`) 100, an AS path that leads directly to the destination and the community c_1. Router R_4 converges to `(200,[`R_3`,`R_1`],[1])`—it prefers the path through R_3 over R_2 because the `lp` of the message along that path is higher.

2.2 Abstractions Can Give Precise Answers about Networks

An analysis of the network in Figure 2 with messages of the form (`lp`, `path`, `comm`) provides precise information about the paths traffic will follow. We know, for example, that R_5 will forward traffic along path $[R_4, R_3, R_1]$. However, such details are unimportant if our goal is simply to validate that all nodes can reach the destination along *some* path. Such reachability questions can usually be answered precisely despite propagating far less information about routes.

Consider instead messages of the form `(R, comm*)`, where the symbol `R` in the first field indicates reachability. If a router has selected any such message, the destination is reachable. The second

[2] In other words, this network is using eBGP everywhere and does not use iBGP. But iBGP can be similarly modeled (§6).

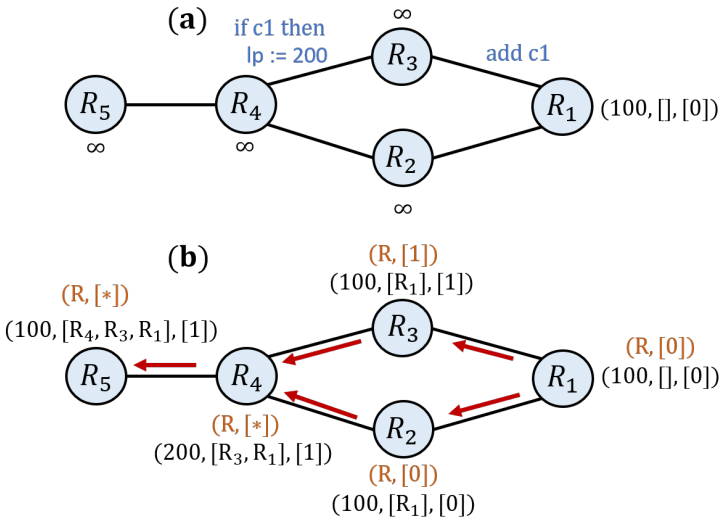

Fig. 2. Example BGP network. (a) The network initial state; non-default transfer functions shown above edges (R_1,R_3) and (R_3,R_4), (b) the stable state for a concrete execution and an abstract execution (orange).

field is again a vector of communities, but this time each community may be 0 (definitely absent), 1 (definitely present) or $*$ (unknown). These messages are significantly more compact because they do not have `localpref` or path.

Figure 2(b) also presents the result of analyzing the network using these abstract messages. The final state is shown in orange. This abstract analysis cannot model the effect of assigning the local preference along the (R_3,R_4) edge. Consequently, it cannot determine whether the message from R_3 (with community c_1) should be preferred over that from R_2 (without c_1). This uncertainty is represented at R_4 using the message `(R,[*])`, indicating it has a route to the destination which may or may not carry community c_1. When the analysis completes, all routers have selected *some* message, and we can conclude that all routers have a path to the destination. If our goal is to guarantee reachability, this abstract analysis for this network yields precise answers.

Of course, such abstract analysis will not always produce precise answers. Suppose the edge between R_4 and R_5 drops messages without c_1 attached. Then, for soundness, the message `(R,[*])` must be dropped when it is transmitted between R_4 and R_5 because the $*$ indicates c_1 may or may not be present. Upon termination, the analysis will report that it cannot guarantee R_5 can reach the destination. However, R_5 can actually reach the destination, because R_4 prefers the message with the community attached.

We have found that, in practice, policies that lose precision are uncommon in real networks (§8), even for simple abstractions. Our example of *path-sensitive* routing, where messages along different paths are modified differently in a way that matters downstream, is uncommon because it decreases fault tolerance. A fault along the R_1, R_3, R_4 path will unnecessarily disconnect R_5 from R_1. Another reason of such a policy being uncommon is that there is a simpler implementation, *e.g.* not exporting the route from R_1 to R_2 in the first place (R_4 would learn the route `(R,[1])`).

2.3 Abstractions Improve Performance of Network Analysis

There are several reasons abstraction can improve the performance of network analysis dramatically:

(1) Abstract messages use less memory and can be processed more efficiently.

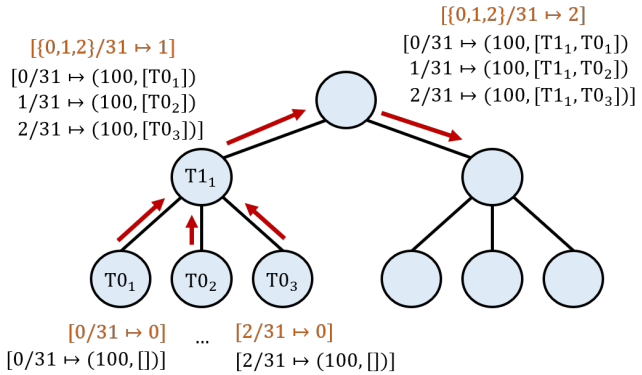

Fig. 3. Example of a data center running eBGP where abstraction improves simulation performance.

(2) Abstraction reduces the number of states a router can inhabit. Hence, analysis requires fewer iterations to converge. For example, BGP can explore $O(2^n)$ paths before converging [Fabrikant et al. 2011], but a reachability abstraction will converge in $O(n)$ iterations (§6.3).
(3) Abstraction enables collapsing of concrete routing messages for many destinations into a single abstract message, which allows for simultaneous, bulk processing of those destinations.

As an example of the last point, consider the datacenter in Figure 3. It runs BGP as before, though it does not use communities and has multiple destinations. To model the data center, routing messages are represented as finite maps that take destination subnets to routes, which are (local preference, AS path) pairs. The figure shows the top-of-rack router $\texttt{T0}_1$ advertising the destination prefix 10.0.0.0/31 (written as 0/31) with local preference 100 and the empty AS path.

A concrete simulation might proceed with $\texttt{T0}_1$, $\texttt{T0}_2$, and $\texttt{T0}_3$ advertising their prefixes to $\texttt{T1}_1$. When $\texttt{T1}_1$ next propagates the routes it learned from the three T0 switches to the spine, it will send the route for each destination prefix separately, and receiving routers will process each message separately. While this processing may appear negligible, as the size of a data center grows, so too do the number of prefixes and processing edges. If the data center has a shortest path routing policy, d destination prefixes, and e edges, then computing the final network state will take $O(tde)$ operations, where t is the time it takes to process a single destination at a router (typically constant)—each prefix will propagate across all edges. For a fattree,[3] a common topology in data centers, this results in $O(n^2 * \sqrt{n})$ operations (and memory) for n devices [Dimitropoulos and Riley 2006].

Now consider an abstract analysis in Figure 3 (orange) that replaces the AS path with an integer representing the path length and increments the length at each hop. When $\texttt{T0}_1$, $\texttt{T0}_2$, and $\texttt{T0}_3$ advertise their routes to $\texttt{T1}_1$, each route advertisement is the same (length 1) and thus can be combined as shown. When $\texttt{T1}_1$ advertises its routes further, recipients can process all destinations simultaneously by simply bumping the path length to 2. If we process nodes in a breadth-first manner (see more about processing order in §6.3), this analysis will complete in $O(te)$ steps. All routes from all destinations at top-of-rack devices will propagate up to the data center spine layer, which learns of all destinations at once, and then propagates back down.

Although it may appear that the time to process messages (t) is now proportional to the number of destinations in the set (d), there are many efficient data structures for processing sets efficiently

[3]Fattrees generalize ordinary tree-shaped graphs such that, instead of just one link from child to parent, there are multiple links from a child to different parents. The additional links allow networks to tolerate some number of faults before becoming disconnected. Fattree topologies are also called Clos networks, after Charles Clos, who formalized them in the 1950s [Al-Fares et al. 2008; Clos Network 2019].

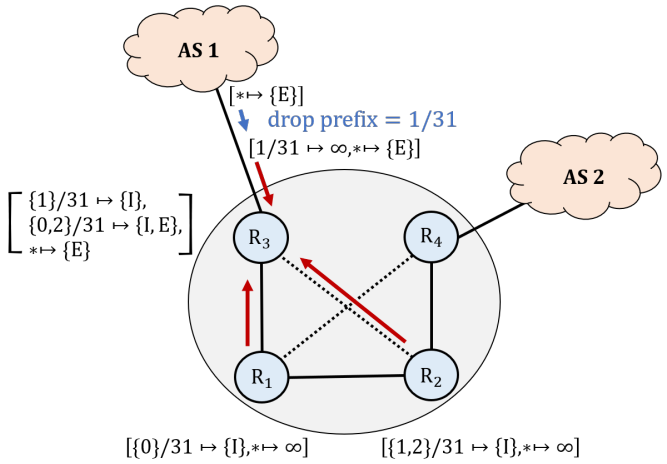

Fig. 4. Example of a new BGP hijacking analysis. Dashed lines are iBGP connections.

(*e.g.*, with union and intersection that are sublinear in d when the sets are mostly disjoint or overlapping [Liljenzin 2013]). For example, for fattrees, the complexity appears to be closer to $O(n*\sqrt{n})$ in practice since destinations advertised by different top-of-rack routers are usually disjoint. Indeed, recall Figure 1 from the introduction. It showed the scaling trends of the Batfish [Fogel et al. 2015] concrete simulator compared to the simple abstract reachability analysis on variable sized fattrees using an efficient set representation (§6.2). Hence, the empirical results back up our complexity analysis: *As topology size increases, the time for concrete vs abstract simulation diverges due to asymptotic differences in complexity.*

2.4 Abstractions Enable New Network Analyses

Abstract analysis in networks can represent messages in unusual ways, which can then enable novel analyses. Consider the wide area network with external peers in Figure 4. Internally, routers again advertise prefixes and the goal is to ensure that all such destinations are reachable. However, a peer might advertise the same prefixes and if there are gaps in BGP filtering (*i.e.*, gaps in the router filters at a network border that block undesirable incoming messages), traffic may route to the peer instead of internal nodes depending on cost. In particular, it will do so if the path through the peer is shorter than the internal path. Unintentional redirection through a peer is known as BGP hijacking and is often caused by accidental actions by a neighboring network.

Using a new abstract domain, we can construct an analysis to efficiently detect potential hijacking attacks of this kind. In particular, we can perform an analysis where we now keep some information about the BGP AS path. Specifically, we abstract the AS path to a boolean value that simply represents whether the origin is internal (`I`) or external (`E`). Abstract routing messages are maps that take prefixes to either ∞ (no route) or a set of internal/external tags. For example, if a route is marked with the set `{I,E}`, it means the final route could either be from an internal device, or from an external peer. To simplify the presentation, we write maps with a default ($*$) key that applies to every other prefix not explicitly written in the map.

In the example, there is a BGP filter to the peer (AS 1) that drops routes for prefix `1/31` but allows them through any other prefixes. To initiate a hijacking analysis, we begin in a state where internal devices map the prefixes they originate to the value `{I}` to mark the route as internal. We assume a peer, in the worst case, will send every possible prefix, so we use map `[* ↦ {E}]`. Of course, one can choose different initial values to make more specific assumptions about the peer's

behavior. The figure shows a single iteration of the analysis focused on router $\mathtt{R}_3$. The peer filter is applied to the external route, resulting in a new route that reflects the dropped prefix: `[1/31` $\mapsto \infty$ `,*` $\mapsto$ `{E}]`. When the internal routes from $\mathtt{R}_1$ and $\mathtt{R}_2$ and those from the external peer are merged, the result is the larger map shown by $\mathtt{R}_3$. The resulting state indicates that the `1/31` prefix can not be hijacked, while it may be possible for peers to hijack the `0/31` and `2/31` prefixes. The analysis would continue until a fixed point is reached.

3 NETWORK MODEL

To reason about the correctness of abstract analyses, we must define a network model and its routing semantics. To this end, we first briefly review routing algebras [Sobrinho 2005], an algebraic model of distributed routing protocols. We then take inspiration from previous work [Daggitt et al. 2018b] to define an asynchronous routing semantics for these algebras. In §3.3, we build on these technical definitions and formally connect *abstract* routing algebras to their concrete counterparts.

3.1 Routing Algebras

A routing algebra defines a distributed routing protocol as a tuple $(S, \oplus, F, 0, \infty)$, where

- S is a set of routes.
- $\oplus : S \to S \to S$ is a merge function that combines two different routes (e.g., from two different peers) by choosing the better route. Here "better" is defined from the perspective of the network operator—a better route may be shorter, have less cost financially, have lower latency or higher capacity. In general, (concrete) routing protocols typically select one of the inputs to $\oplus$ as the better route. However, for modeling purposes, we allow $\oplus$ to combine its inputs in any way as long as it is commutative and associative.
- F is a family of transfer functions, where each function $f \in F : S \to S$ transforms a route as it is processed along an edge (*e.g.*, by adding an edge cost). We use the notation f_{ij} to refer the transfer function between routers i and j.
- 0 is the initial route, and
- ∞ is the invalid route.

In the simplified eBGP example of §2, the set of routes S includes $(\mathbb{N} \times \text{list}(V) \times \text{list}(\{0,1\})) \cup \{\infty\}$. Each valid route is a tuple of `localpref`, `path` and `communities`, where V is the set of network nodes. The route ∞ represents the lack of a route—it is used when a route from a neighbor is filtered, for example.

For the merge function $\oplus$, we let $a \oplus \infty = \infty \oplus a = a$. In other words, a router always prefers a valid route over no route at all. Moreover, if $a = (l_1, p_1, c_1)$ and $b = (l_2, p_2, c_2)$, the merge function for this simplified version of eBGP is defined as:

$$a \oplus b = \begin{cases} a & \text{if } l_1 > l_2 \text{ or } (l_1 = l_2 \wedge |p_1| < |p_2|) \\ b & \text{otherwise} \end{cases}$$

In other words, routes are ordered lexicographically, first by local preference and second by the length of their paths.

In BGP, user can customize their transfer functions using BGP configurations. Hence, each transfer function $f \in F$ (one per edge) applies user-defined policy before appending the current hop to the BGP path. The initial route (0) is the route `(100,[],[0,0,...])` with an the empty path and all communities set to 0.

3.2 Asynchronous Semantics

While a routing algebra defines the basic operations routers perform to modify messages, it does not say how such messages are processed on a given topology. Each device inspects the routing messages from its neighbors after applying neighbor-specific transfer functions $f \in F$, and then merges the messages using $\oplus$, until the network reaches a fixed point. This computation is complicated by the fact that messages are exchanged asynchronously and can be reordered. We define an asynchronous network semantics inspired by the work of Daggitt *et al.* [Daggitt et al. 2018b]. It uses a linear notion of time, which is captured by the set of time steps $\mathcal{T}$.

Definitions. Given a routing algebra $A = (S, \oplus, F, 0, \infty)$, we give a semantics with respect to a *network instantiation* (π) which is a tuple $\pi = (V, E, I, X, \tau, \omega)$, where:

- V is a set of vertices, and $E \subseteq V \times V$ is a set of edges in the network topology.
- I is an *initial* routing state (the state for a fresh network), where $I_j \in S$ denotes the initial route for router j. I_j is typically 0 if j initially announces the destination, otherwise ∞.
- X is the *starting* routing state from which execution begins (not necessarily the same as I). For example, X may represent a partially converged state.
- $\tau : \mathcal{T} \to 2^V$ is an asynchronous time schedule that maps a time step to the set of vertices that process routes at that time step.
- $\omega : \mathcal{T} \to V \to V \to \mathcal{T}$ is a data flow function where $\omega(t, i, j)$ gives the time step where the information used at vertex i at time t was sent by vertex j. We require that $\omega(t, i, j) < t$, *i.e.*, disseminating a routing advertisement takes a non-zero amount of time.

We now define a semantics σ with respect to an algebra A. The semantics gives the state of the network after t time steps for an instantiation π. The state of the network at time t for node i with instantiation $\pi = (V, E, I, X, \tau, \omega)$ is defined by the following:

$$\sigma^0(\pi)_i = X_i$$

$$\sigma^t(\pi)_i = \begin{cases} \sigma^{t-1}(\pi)_i, & \text{if } i \notin \tau(t) \\ I_i \oplus \left(\bigoplus_{(i,j) \in E} f_{ij}(\sigma^{\omega(t,i,j)}(\pi)_j) \right), & \text{if } i \in \tau(t) \end{cases}$$

After zero time steps, the state at node i is given by the starting state in X. To find the state after time step t for node i, it checks if i is currently scheduled to process an update ($i \in \tau(t)$). If not, then the state at node i is the same as at time step $i - 1$. Otherwise, node i will update its state by merging the last routes it has learned from each neighbor under schedule ω. (Recall that f_{ij} refers to the transfer function corresponding to the edge (i, j).) Note that node i updates its state by merging messages from each neighbor with its initial state I_i rather than its previous state $\sigma^{t-1}(\pi)_i$. Consequently, if a neighbor sends route a now, and then a different route b later, when i receives b, it will forget that the neighbor ever announced a.

3.3 Routing Algebra Abstraction

An abstraction of a routing algebra $A = (S, \oplus, F, 0, \infty)$ is yet another routing algebra $A^\sharp = (S^\sharp, \oplus^\sharp, F^\sharp, 0^\sharp, \infty^\sharp)$, equipped with a pair $(\alpha, \sqsubseteq^\sharp)$ relating the two algebras. Here, $\alpha : S \to S^\sharp$ is an abstraction function that maps every concrete route to an abstract route, and $\sqsubseteq^\sharp$ is a partial order over $S^\sharp$.

BGP Example. Consider again the eBGP example from §2. Abstract routes $S^\sharp$ are of the form `(R,comms)` to indicate a reachable destination with communities `comms`. Community tags can be `0` (absent), `1` (present), or $*$ (uncertain). First we define the merge function $\oplus^\sharp$ for the abstract domain: $a \ \oplus^\sharp \ \infty^\sharp = \infty^\sharp \ \oplus^\sharp \ a = a$, and $(R, C_1) \ \oplus^\sharp \ (R, C_2) = (R, C_1 \ \oplus^\sharp \ C_2)$. Here, the community vectors

	Property	Definition
(1)	$\oplus, \oplus^\sharp$ commutative	$c \oplus d = d \oplus c$
(2)	$\oplus, \oplus^\sharp$ associative	$c \oplus (d \oplus e) = (c \oplus d) \oplus e$
(3)	$\oplus^\sharp$ monotone	$a \sqsubseteq^\sharp c \wedge b \sqsubseteq^\sharp d \implies a \oplus^\sharp b \sqsubseteq^\sharp c \oplus^\sharp d$
(4)	$\oplus, \oplus^\sharp$ sound for α	$\alpha(c \oplus d) \sqsubseteq^\sharp \alpha(c) \oplus^\sharp \alpha(d)$
(5)	$F, F^\sharp$ sound for α	$\alpha(f(c)) \sqsubseteq^\sharp f^\sharp(\alpha(c))$
(6)	$F^\sharp$ monotone	$a \sqsubseteq^\sharp b \implies f^\sharp(a) \sqsubseteq^\sharp f^\sharp(b)$

Fig. 5. Summary of required abstraction conditions for routing algebras.

are merged pointwise with: $(C_1 \ \oplus^\sharp \ C_2)_i = C_{1_i} \ \oplus^\sharp \ C_{2_i}$ and individual communities are merged as:

$$c_1 \ \oplus^\sharp \ c_2 = \begin{cases} c_1 & \text{if } c_1 = c_2 \\ * & \text{otherwise} \end{cases}$$

Merging `(R,[0,1])` $\oplus^\sharp$ `(R,[1,1])` results in `(R,[*,1])`. Now, we can relate this abstract routing algebra to the concrete BGP algebra by defining a suitable abstraction function α. We define

$$\begin{aligned} \alpha(\infty) &= \infty^\sharp \\ \alpha((lp, path, C)) &= (R, C) \end{aligned}$$

The abstraction maps ∞ to $\infty^\sharp$, replaces the BGP local preference value and path with a reachability marker R, and keeps the communities C intact. The partial order $\sqsubseteq^\sharp$ over $S^\sharp$ is defined as follows.

$$(R, C_1) \sqsubseteq^\sharp (R, C_2) \iff C_1 \ \oplus^\sharp \ C_2 = C_2$$

The abstraction of no route ($\infty^\sharp$) is incomparable to the abstraction of any valid route (R, C).

4 SOUNDNESS

In this section, we prove two theorems concerning the soundness of abstract routing algebras with respect to their concrete counterparts. These soundness results depend upon a set of algebraic conditions laid out in §4.1.

Our first theorem states that if one fixes a concurrent schedule $s = (\tau, \omega)$ then an abstract execution using schedule s is a sound overapproximation of a concrete execution with the same schedule s. Using this theorem, we know a single execution of the abstract algebra is sound with respect to an execution of the concrete algebra that uses the same schedule, but we do not know if the execution of the abstract algebra is sound with respect to other executions of the concrete algebra. Hence, if an abstract algebra has many solutions, we must execute it many times—at least one per solution—in order to cover all possible concrete solutions. Moreover, because we know of no method that can predict which interleavings of messages will give rise to new and different solutions, we may have to consider all possible interleavings, which is prohibitively expensive. In general, abstract interpretation of highly concurrent systems (*e.g.*, with many threads) can lead to a combinatorial explosion in complexity [Monat and Miné 2017].

Fortunately, in practice, the situation is promising. Routing algebras, unlike most concurrent systems, typically have a single unique solution. For instance, Lopes and Rybalchenko observed that Microsoft's networks have sufficient monotonicity properties to imply convergence to a unique solution [Lopes and Rybalchenko 2019]. Likewise, Gao and Rexford famously proved that the internet's economic structure disincentivizes network operators from configuring their networks to generate unstable behavior [Gao and Rexford 2000]. Last, our discussions with network operators have not turned up situations in which these operators fear non-convergence or multiple solutions.

Conditions for convergence and uniqueness of solutions are well-understood and have been studied extensively [Daggitt et al. 2018a; Sobrinho 2005]. Hence, even in the unlikely case that the concrete algebra has multiple solutions, we can craft abstract routing algebras that (provably) have a single solution. Consequently, any single abstract execution will suffice to soundly overapproximate all concrete executions. The specific abstractions we suggest in this paper have that property (See §6.1) as they abstract features such as the BGP local preference that can lead to multiple solutions.

Hence, our second theorem states the strong property that one wants and that applies in practice: If the abstract algebra has a unique solution, then it is sound with respect to *all* concrete simulations (even if the concrete system has many solutions). Consequently, a single run of the abstract simulator, with any concurrent schedule suffices.

We state and prove our two theorems in §4.2 from first principles. In §4.3, we discuss the relationship to the standard Galois connection construction.

4.1 Abstraction Conditions

Figure 5 lists sufficient conditions for an abstract algebra to be sound with respect to a concrete algebra. Conditions 1-4 constrain the behavior of the merge functions. §3.2 The first two conditions require that both $\oplus$ and $\oplus^\sharp$ are commutative and associative. Condition three requires that $\oplus^\sharp$ is monotone, and the fourth condition relates $\oplus$ and $\oplus^\sharp$ by requiring $\oplus^\sharp$ to overapproximate $\oplus$ with respect to the abstraction α. Conditions five and six are fairly standard. They ensure that applying an abstract transfer function $f^\sharp$ from a single neighbor should overapproximate the behavior of applying the concrete transfer function f from the same neighbor, and the abstract transfer function $f^\sharp$ should be monotone respectively.

4.2 Soundness Theorems

To prove that the abstraction conditions from §4.1 suffice to ensure soundness, we must first lift the notions of abstract ordering ($\sqsubseteq^\sharp$) and abstraction (α) from routes to routing states (vectors of routes, one per node).

Definition 4.1. *Given abstract routing states X, Y, we define $X \sqsubseteq^\sharp Y$ iff for each node i, $X_i \sqsubseteq^\sharp Y_i$. Similarly, we lift α to routing states pointwise by defining $\alpha(X)_i = \alpha(X_i)$.*

Now, given a fixed schedule (τ, ω) we show that an abstract execution is a sound overapproximation of the concrete execution with the same schedule.

Theorem 4.1. *Consider algebra A with semantics σ and abstract algebra $A^\sharp$ with semantics $\sigma^\sharp$, where A and $A^\sharp$ are related by sound abstraction $(\alpha, \sqsubseteq^\sharp)$. For all times t, concrete instantiations $\pi = (V, E, I, X, \tau, \omega)$, and abstract instantiation $\pi^\sharp = (V, E, \alpha(I), \alpha(X), \tau, \omega)$ we have $\alpha(\sigma^t(\pi)) \sqsubseteq^\sharp \sigma^{\sharp t}(\pi^\sharp)$.*

Proof. By strong induction on the time t. See the Appendix for details. □

Theorem 4.1 shows that the abstract analysis is sound for any fixed schedule. If the abstract algebra is guaranteed to converge to a *unique* solution, then we can show that its answer is sound for *any* possible concrete schedule.

Definition 4.2. *We say algebra A with semantics σ is* uniquely converging *for topology (V, E), and states (I, X) if there is a stable state X^* such that for all schedules (τ, ω), there exists a convergence time t such that for all k, $\sigma^{t+k}(\pi) = X^*$ for $\pi = (V, E, I, X, \tau, \omega)$.*

Theorem 4.2. *Consider algebra A with semantics σ, and abstract algebra $A^\sharp$ with semantics $\sigma^\sharp$, related by sound abstraction $(\alpha, \sqsubseteq^\sharp)$. If $A^\sharp$ is uniquely converging for topology (V, E) and states*

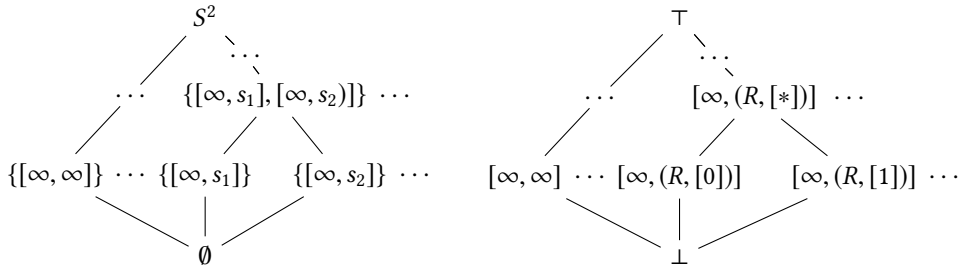

Fig. 6. The concrete (left) and abstract (right) routing state lattices from Figure 2. Concrete route $s_1 = (200, [R_3, R_1], [0])$ and $s_2 = (200, [R_3, R_1], [1])$.

$(\alpha(I), \alpha(X))$, then for all schedules (τ_1, ω_1) and (τ_2, ω_2), there exists a time t such that for all k, $\alpha(\sigma^{t+k}(\pi)) \sqsubseteq^\sharp \sigma^{\sharp t+k}(\pi^\sharp)$ for $\pi = (V, E, I, X, \tau_1, \omega_1)$, and $\pi^\sharp = (V, E, \alpha(I), \alpha(X), \tau_2, \omega_2)$.

There are a few points worth noting. First, unlike previous work [Daggitt et al. 2018b], this theorem does not identify conditions under which a routing algebra converges uniquely. But *if* an abstract algebra does, then any single abstract execution is sound with respect to *all* concrete executions—a powerful result. Second, a concrete algebra need not converge uniquely. For instance, a BGP configuration may not converge uniquely, perhaps due to inconsistent use of local preference, but our abstractions of BGP (those that elide local preference, for instance) could!

The theorems above are general in that they consider abstract/concrete executions for a fixed schedule and for all schedules, respectively. They can be applied in various settings for control plane analysis, such as:

- For concrete simulation of control planes as in Batfish [Fogel et al. 2015], Theorem 4.1 guarantees that an abstract simulation is sound with respect to a concrete simulation.
- In addition, if an abstract algebra converges uniquely, then a single abstract simulation performs verification, since Theorem 4.2 ensures soundness over *all* schedules.
- For control plane verification, our abstractions can be used by a tool like Minesweeper [Beckett et al. 2017]. Since Minesweeper implicitly searches over stables paths due to all possible schedules, our abstractions could improve performance while guaranteeing soundness (*i.e.*, no bug in an abstract model implies no bug in the concrete model). These can also be used in counterexample guided abstraction refinement [Clarke et al. 2000].
- Verification based on model checking (*e.g.*, Plankton [Prabhu et al. 2017]) can check invariants over *transient* states in a network, *i.e.*, before the network reaches convergence. Our abstractions would enable checking executions over abstract transient states. Again, uniquely converging abstract algebras would ensure soundness over all possible schedules.

4.3 An Alternative Development using Galois Connections

One may wonder what is the relationship between Theorems 4.1, 4.2 and the standard theory from abstract interpretation. An alternative way to formulate the problem would be to define a concrete domain for a network with n nodes as the powerset of routing states $\mathcal{P}(S^n)$ and the abstract domain as abstract routing states $S^{\sharp n}$. A synchronous semantics that updates the route at every node for each time step could be represented as a function $\sigma : S^n \to S^n$. For soundness one would assume a Galois connection $(\mathcal{P}(S^n), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (S^{\sharp n}, \sqsubseteq^\sharp)$ between the concrete and abstract domains, as well as sound abstract transformers. Soundness of the abstract transformers is related to conditions 3-6 in Figure 5, which ensure soundness of merge and transfer functions independently. Assuming the

	Object	Abstraction	Concrete example c	Abstract example a ($\alpha(c) = a$)
1.	Path cost	Reachable	`5`	`R`
2.	Path vector	Path length	`[100, 200, 301]`	`3`
3.	Path vector	Neighbor	`[100, 200, 301]`	`{301}`
4.	Path vector	Waypoints	`[100, 200, 301]`	`{100, 301}`
5.	Path vector	Reg exp.	`[100, 200, 301]`	`"(1|2|3)(0)(0|1)"`
6.	Path vector	Origin	`[100, 200, 301]`	`{I,E}`
7.	Tag set	3-valued logic	`[1,0,1]`	`[1,*,*]`
8.	Tag set	Powerset	`{1:50,1:67}`	`{{1:49},{1:50,1:67}}`
9.	BGP local pref	Cost: `{C,P,R}`	`250`	`P`
10.	Any field	Drop field	`(100, 80, [100])`	`(100,80)`
11.	IP address	Ternary abs.	`10001101`	`10***101`
12.	IP address	Interval abs.	`00001101`	`[0,15]`
13.	Available paths	Set of edges	`{[R1, R3],[R1, R2, R3]}`	`{(R1,R3),(R1,R2),(R2,R3)}`
	Combinators			
14.	Destination	Map	$[164/8 \mapsto \infty, 170/8 \mapsto 3]$	$[164/8 \mapsto \infty, 170/8 \mapsto R]$
15.	Path	Product	`[100, 200, 301]`	`(3, {I,E})`

Fig. 7. Examples of abstractions and combinators for routing.

concrete algebra (σ) is monotone, then the standard abstract interpretation theory would apply. Figure 6 shows an example of the concrete and abstract lattices corresponding to the algebra used in the example from Figure 2 (restricted to a network with 2 nodes for simplicity).

There are several notable differences between this "more standard" formulation and that of §4. First, unlike in abstract interpretation of traditional program where a collecting semantics overapproximates the set of concrete values reachable at each program location by merging results from different program paths, in routing algebras there is no equivalent notion of branching. As a result, the full machinery of abstract interpretation is not needed – we do not require a complete lattice with a join ($\sqcup$) defined. In particular, one will never obtain values that extend above the second bottom level of the concrete lattice for any concrete execution. Second, the formulation of abstractions in §4 as a routing algebra makes it easy to leverage existing results from the routing algebra literature on unique convergence to obtain soundness for all possible asynchronous schedules without having to account for asynchrony in the abstraction itself (Theorem 4.2). This both simplifies the theory and improves performance since the implementation can fix any particular schedule. Further, defining an abstraction as an abstract routing algebra has the practical advantage enabling reuse of existing routing simulation engines, which we take advantage of in §7.

5 ABSTRACTIONS

While we showed example abstractions earlier, we now showcase a rich collection of routing abstractions. Several of the abstractions are similar to ideas in previous work, while many others are completely new in the context of network verification. Figure 7 shows an overview. The columns denote: (1) the object being abstracted, (2) the idea behind the abstraction, (3) an example of a concrete instance of the object, and finally, (4) the corresponding abstract instance for the concrete example. The combinators in the last two rows help compose a richer abstraction from simpler ones, where the soundness claims carry through.

Path abstractions. Rather than keeping only information about the existence of a path (`R`), one can keep any number of other pieces of information with tradeoffs in terms of representation size and precision. Several such examples are listed in Figure 7 (Rows 2-5). One example is to abstract a

path as its length. For instance, BGP carries around a path of AS numbers that a route has traversed. In this case, the abstraction function is $\alpha(\text{path}) = |\text{path}|$.

Alternatively, one could track other information about the path such as the potential origin of the path or the particular external neighbors through which the route might have been learned. More generally, one could track a set of waypoints of interest that the route has gone through.

Another interesting abstraction is to encode a path using a regular expression. A regular expression can overapproximate the set (or sequence) of elements in the path. In Row 5, the regex `"(1|2|3)(0)(0|1)"` captures the set `{100,200,301,101,201}`, a superset of the actual concrete values. Regular expressions afford a great deal of flexibility in the granularity to track path elements. For instance, one can use regexes to enumerate all possible values in a path.

Environment. The BGP hijacking example from Figure 4 can be captured with a simple internal/external environment abstraction of the BGP AS path (Row 6).

Tags and bitfields. Protocols often keep a collection of tags or bit fields (*e.g.*, BGP communities). Given a finite set of tags or bits where each value is either 1 or 0, a simple way to abstract these tags is to use a ternary representation (Row 7), where each element is abstracted as either 1, 0, or `*`.

To retain more information when representing concrete tag sets, one can use a powerset abstraction [Filé and Ranzato 1999] (Row 8), which explicitly tracks the set of possible tag sets. The powerset abstraction gives more precision at the cost of representation space. For example, suppose two tags `x` and `y` are only ever set together depending on which route was chosen, and policy downstream applies an action only if the two values are the same. The powerset abstraction retains the set `{[0,0],[1,1]}` and loses less precision downstream. In contrast, the ternary representation would be `[*,*]`.

BGP Local Preference. The BGP local preference is a policy attribute used to hard code preferences for certain routes over others. Local preference is responsible for a number of problems in BGP such as route oscillation [Griffin et al. 2002; Varadhan et al. 1996]. However, BGP preferences are often used in a safe way by ordering routes between customers, peers, and providers. One way to abstract local preference would be to translate the local preference value directly into the a smaller set of finite values corresponding to these preferences (Row 9). For instance it is common for wide-area-networks to assign local preferences between ranges based on cost, *e.g.*, between 100 and 200 might be used for peers (P), while 201 to 300 is used for providers (R).

Decision Variables. Alternatively, one can drop values used only in a protocol's decision process such as local preference, origin-type, and others (Row 10). It may not be important what route is chosen so long as the destination can be reached. Given a route that is a tuple of values, one can project away one or more of the fields.

Destination IP. The destination IP address (Rows 11, 12) can be viewed as a collection of bits, and thus any of the abstractions for collections bits can be used for the IP address. For instance, one can use a ternary representation or a powerset construction. Since IP addresses are often matched by wildcards on the least significant bits, another type of useful abstract may be an interval abstraction (example 12), which abstracts an IP address as a line segment.

Edge abstraction. The BGP protocol maintains a set of concrete paths when used in a multipath setting, for example in production data centers [Lapukhov et al. 2015]. This can lead to a large amount of redundancy in routing tables when many similar, but overlapping, paths exist to the same destination. It is possible to abstract the set of paths, which may contain overlap, into a set of edges (Row 13) thus eliminating the redundancy at the cost of forgetting the exact set of paths.

Keeping the set of edges an advertisement may traverse can also be useful because it provides an abstract view of the "flow" of control plane information in the network. Such information can be used, for instance, to quickly identify the degree of failure needed to disconnect a source from a destination—an insight leveraged in ARC [Gember-Jacobson et al. 2016].

5.1 Map Abstraction

It is well known that one can lift a sound abstraction between concrete values C and abstract values A to functions over values $X \to C$ and $X \to A$. This is often done to capture variable assignment, where the set S may represent a set of variables. This idea is particularly useful for networks, where key-value tables are a recurring data structure. For example, routers will maintain a map from prefix to best route in each protocol.

Such a lifting from values to key-value tables (or more specifically, routes to prefix-route tables) may also be applied to routing algebras. Suppose we have a concrete routing algebra $A = (S, \oplus, F, 0, \infty)$. Given a set X, we can construct a new routing algebra over functions from X to S that abstracts elements pointwise: $\text{Map}(X, A) = (X \to S, \overrightarrow{\oplus}, \overrightarrow{F}, \overrightarrow{0}, \overrightarrow{\infty})$, where:

$$g \overrightarrow{\oplus} g' = \lambda x.\ g(x) \oplus g'(x) \qquad \overrightarrow{0} = \lambda x.\ 0$$
$$\overrightarrow{f}(g) = f \circ g \qquad \overrightarrow{\infty} = \lambda x.\ \infty$$

Given a sound abstraction α between A and $A^\sharp = (S^\sharp, \oplus^\sharp, F^\sharp, 0^\sharp, \infty^\sharp)$, we can define a sound new abstraction $(\overrightarrow{\alpha}, \overrightarrow{\sqsubseteq})$ between $\text{Map}(X, A)$ and $\text{Map}(X, A^\sharp)$ when $\overrightarrow{\alpha}(g) = \alpha \circ g$ and $g_1 \overrightarrow{\sqsubseteq} g_2 \iff \forall x.\ g_1(x) \sqsubseteq^\sharp g_2(x)$.

6 FAST REACHABILITY ANALYSIS

To demonstrate the practical benefits of routing abstractions, we now describe the design and implementation of ShapeShifter, a new tool for reachability analysis. We focus on reachability because it is a key property of interest to network operators [Fayaz et al. 2016; Weitz et al. 2016]. The goal of our analysis is to compute the set of destination prefixes that can be reached from each source, and by extension, which packets can travel between a source-destination pair. We structure the abstract analysis to soundly compute the "must reach" relationship—if the analysis declares reachability between a source-destination pair, then indeed packets can make that journey. The analysis is conservative and may fail to identify some reachable source-destination pairs.

6.1 What to Abstract

The first order of business is to decide what abstractions to use to enable fast analysis while minimizing precision loss for real networks. We made the following decisions based on the characteristics of over a hundred real networks (§8). The exact abstractions may differ for other networks. We defer automatically determining the appropriate abstraction for a given network to future work.

- **Throw away the BGP AS path**. We found that we could throw away the BGP AS path, keeping only a simple reachability marker, without losing much precision. Similarly, we could throw away the IGP (internal) path cost and replace it with a reachability marker. Keeping only a reachability marker simplifies route representation and allows for efficient prefix-set representations.
- **Keep the BGP Origin**. While throwing away the entire AS path often did not result in significant precision loss, we found that keeping the BGP origin allowed for better error messages. Further, it enables efficient all-pairs, all-packets dataplane reachability analysis (§6.6). Specifically, we use a powerset abstraction to keep the set of possible origins (in case of anycast).
- **Keep the BGP communities**. BGP communities are one of the few attributes we found were important to retain since BGP routers often make important filtering decision downstream based

on tags applied upstream, and discarding community information could lead to high precision loss. We found that a simple ternary abstraction often suffices but we use a powerset abstraction since community sets can be represented with little additional cost using BDDs (§6.2).

- **Discard all decision variables**. Since we do not keep track of what path is used, we decided to discard all route fields related to protocol decision processes. This includes fields such as BGP local preference, multi-exit discriminator, path length, administrative distance, and so on.
- **Remember the protocol**. Routers can run more than one protocol, and determining which protocol is used for forwarding (*i.e.*, which route is stored in the forwarding table, or FIB) is complex. We maintain a set of possible chosen protocols: *e.g.*, `{CONNECTED, STATIC, OSPF, BGP, IBGP}`. We found that losing information about exactly which protocol was used to learn a route is often fine in the context of reachability, since it may not affect the presence of a route (*e.g.*, OSPF vs. BGP). On the other hand, it is possible to lose precision since one may not be able to redistribute a route from one protocol to another if the protocol of the chosen route stored in the FIB is not known.
- **Map abstraction**. We use the Map abstraction to soundly lift the aforementioned routes to be functions from destination prefix to abstract route.

An example abstract routing message in the final reachability domain would be `[168/8 ↦ ([0,*,0], {R1}, {BGP})]`, which denotes that the second community could be attached or not on a BGP route originating from R1.

6.2 Prefix Set Representation

As described in §2, simplifying the route representation allows for efficient bulk processing of messages. We use Binary Decision Diagrams (BDDs) [Bryant 1986] as a data structure to efficiently represent, merge, and process entire sets of abstract messages at a time. Since route origins often differ across otherwise similar routes, reducing sharing, we represent information as a map from tuples of prefix and origin to abstract communities and protocols (*e.g.*, `[(168/8,{R1}) ↦ ([0,*,0], {BGP})]`). BDDs can then be used to efficiently represent sets of keys (prefix and origin), while allowing sharing between the mapped values.

6.3 Analysis Time

Abstract analysis time depends on the abstract algebra and the message schedule. For the simplest reachability abstraction, each device can only change its state at most once, from ∞ to R, but never the other way. As a result, the analysis must converge in $O(n)$ iterations for a fixed destination, regardless of the network topology and policy. In contrast, the concrete algebra can take many more iterations [Fabrikant et al. 2011; L. Daggitt and Griffin 2018].

Equally important, the order in which the routers process routes (the schedule from §3) can have a large impact on the analysis time. We use a global priority queue of routes (pairs of prefix and route), where entries in the priority queue are ordered according to $\sqsubseteq^\sharp$. The intuition, also exploited in work on fast BGP simulation [Lopes and Rybalchenko 2019], is that it is beneficial to propagate routes that are "preferred" first since propagating less preferred routes first only to have them be overwritten later can lead to redundant processing.

This strategy means: (1) routes with lower path cost should be processed first, (2) routes with better administrative distance, followed by BGP local preference, etc. should be processed first (if modeled), (3) routes with more unknown (*) community values should be processed first, (4) routes with a larger edge set should be processed first, and so on.

6.4 Incremental Merging

Applying the algebraic semantics in §3 requires that, at each step of the computation, a node will update its current best route by applying $\oplus$ to all of its current routes from neighbors. However, this is not always necessary. Suppose node i currently has the route $X_i = N_1 \oplus \ldots \oplus N_k$, learned from neighbors. Now suppose that neighbor j sends a new route N'_j, which replaces N_j. If $N_j \oplus N'_j = N'_j$, then since $\oplus$ is commutative and associative, it allows for an incremental update since $X_i \oplus N'_j = N_1 \oplus \ldots \oplus (N_j \oplus N'_j) \oplus \ldots \oplus N_k$ which is equal to the new desired value $N_1 \oplus \ldots \oplus N'_j \oplus \ldots \oplus N_k$. This condition holds if the neighbor has yet to send a route ($N_j = \infty$), which is often the case. It also holds when $\oplus$ is *selective*—that is, when $a \oplus b = a$ or $a \oplus b = b$ (as opposed to $a \oplus b$ being some combination of the information in both a and b, as would be the case if $\oplus$ was set union, for instance). Only when these conditions do not hold, do we recompute X_i from scratch.

6.5 Other Modeling Concerns

In addition to finding efficient data structures to represent and process routes, we have to address several real-world implementation details.

Multiple protocols. While much of the discussion of routing algebras has been in the context of a single routing protocol and/or destination, modeling multiple protocols and their interactions is possible. For instance, to model static routes, OSPF and BGP, we create a new algebra of the form $A_{STATIC} \times A_{OSPF} \times A_{BGP} \times A_{FIB}$, where each individually represents the algebra for its own protocol, and A_{FIB} stores the best route among all protocols. Protocols are merged as: $(s_1, o_1, b_1, f_1) \oplus (s_2, o_2, b_2, f_2) = (s', o', b', f')$ where each protocol is merged separately: $s' = s_1 \oplus_s s_2$, $o' = o_1 \oplus_o o_2$, $b' = b_1 \oplus_b b_2$, and the chosen route is picked according to the minimum administrative distance configuration parameter: $f' = (STATIC, s') \oplus_{ad} (OSPF, o') \oplus_{ad} (BGP, b')$. This modeling approach can easily model interactions among protocols. For instance, BGP will only export routes that exist in the FIB, which makes redistributing routes between different protocols [Redistributing Routing Protocols 2012] simple to model as part of the transfer function.

iBGP. Connectivity between BGP-configured border routers in the same autonomous system is typically achieved using the iBGP protocol, which can be modeled with routing algebras [Griffin and Sobrinho 2005]. For eBGP routes to be delivered between iBGP peers, the iBGP-configured next hop address must be reachable in the underlying IGP. Since we are creating a domain for reachability, dealing with this complex behavior is easy—we are already computing "must reach" sets of routes for all destinations, so we simply look to see if the iBGP next hop address is reachable when processing an eBGP route.

6.6 Dataplane Reachability

The reachability analysis described thus far computes a set of "must reach" control plane routes, but it says nothing yet about how packets are forwarded in the data plane. Because a packet with a fixed destination IP such as `10.0.15.1` may match multiple routes with different prefixes like `10.0.15.0/24` and `10.0.0.0/8`, routers use longest matching prefix (/24 in this case) to decide which route to use.

Intuitively, it is easy to infer reachability facts about the data plane from the control plane. For instance, if we have a final route: `[168.0.0.0/8 ↦ ([0,0,0], {R1}, {BGP}), 168.2.0.0/16 ↦ ([0,0,0], {R1,R2}, {BGP})]`. We can easily determine that packet matching `168.2.*.*` will either end up at `R1` or `R2`, and any other packet matching `168.*.*.*` will end up at `R1`.

Going from the set of "must reach" routes to the set of "must reach" dataplane packets using BDDs works as follows. We iteratively filter the set of final routes for routes with length k, where k

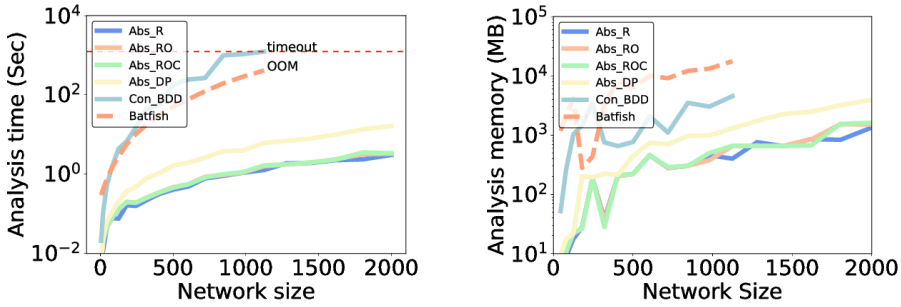

Fig. 8. Network size vs. analysis time (left) and peak memory usage (right), for different abstractions.

starts at 32 and ends at 0. From these routes, we then existentially quantify away the prefix length and the last $32 - k$ bits of the prefix. This converts the prefix in the route to a symbolic set of packets representation, and we accumulate this set of packets matched so far. At each step, we must then remove any values in this set that are already matched by any route with prefix length $k' > k$. This is done using set difference (BDD negation). The final result is a new BDD representing a map from packet to route. Determining the set of packet headers that will reach different locations is then simple since we stored the origin as part of the route. Interestingly, this allows the analysis to reuse almost all the work done in executing the control plane to give a sound answer to the all pairs, all packets data plane reachability problem (*i.e.*, if the analysis says that a packet is "reachable", then it really is reachable).

Access control lists. It is possible to configure access control lists (ACLs) on individual router interfaces that can block different types of packets (*e.g.*, do not allow packets with destination port 53 unless the protocol is UDP). While ACLs do not affect control plane reachability, they do affect data plane reachability. To address this, when checking data plane reachability, we track the set of packet headers that might be blocked by ACLs. When a set of routes is propagated from one router to another, if there are ACLs configured between the devices, then we convert the set of destinations to a header space (again using BDDs) and intersect it with the header space represented by the ACLs (also as BDDs). We then accumulate the set of potentially blocked packets due to ACLs.

7 IMPLEMENTATION

We implemented ShapeShifter as a simulation engine following the network semantics laid out in §3. ShapeShifter uses Batfish [Fogel et al. 2015] to parse network configurations into a vendor-neutral data model, and then operates over this model. The implementation uses BDDs to represent, batch, and process a set of routing messages at a time (§6.2). ShapeShifter currently implements several reachability abstractions (§8.2) with varying degrees of precision.

8 EVALUATION

The primary goals of our experiments are to illustrate tradeoffs between performance and precision for abstract routing analysis and to find abstractions that work well for a broad class of industrial networks. Our goal is not an exhaustive exploration of the space of all possible abstractions. More specifically, we are interested in (1) how analysis time scales with network size, (2) whether the precision of our abstractions suffices for real networks, and (3) whether abstract analyses can scale well for real networks.

8.1 Networks Considered

We consider two types of networks. First, we use generated fattree topologies [Al-Fares et al. 2008] that run eBGP with shortest path routing policy, to evaluate how well different abstractions perform as network size grows in a controlled manner. Second, we use 127 production networks from a large cloud provider. The production networks range in size from several devices to many hundreds of devices, and from thousands to hundreds of thousands of lines of configuration. They use a variety of protocols and features including, but not limited to, static routes, route redistribution, BGP local preference, communities, regular expression filters, private and public AS numbers, and ACLs. Most specialized analyses [Fayaz et al. 2016; Gember-Jacobson et al. 2016; Weitz et al. 2016] are unable to handle all of these features. For confidentiality reasons, we do not report specifics of routing policies or network topologies.

8.2 Scaling Trends (Synthetic Networks)

To observe the effect of topology size on analysis performance, we study several abstractions using the fixed shortest path policy with the generated networks. Figure 8 shows the scaling trends of different abstractions that use BDDs and of Batfish. It plots the median analysis completion time (left) and maximum memory usage (right) observed over 10 runs. **Abs_R** is the simplest reachability abstraction (§2.3), **Abs_RO** additionally tracks the BGP origin, **Abs_ROC** also tracks BGP communities using the powerset abstraction, **Abs_DP** is Abs_ROC but also computes all pairs, all packets data plane reachability, **Con_BDD** performs no abstraction but still represents routes using BDDs. Batfish performs concrete analysis using a standard representation. Abs_ROC corresponds to the reachability analysis described in §6.1. Experiments were performed on PC with a 6-core, 3.6 GHz Intel Xeon processor and 32 GB of RAM. We use a 25 minute analysis timeout and abort runs that exceed memory.

From the left graph, we observe that all reachability abstractions (Abs_R, Abs_RO, Abs_ROC, and Abs_DP) are cheap relative to concrete simulation (Con_BDD, Batfish). The reachability abstractions complete in a couple of seconds for the a 2000 router data center, while concrete simulation with either Con_BDD or Batfish take orders of magnitude longer for smaller networks. Above 1200 routers, Batfish runs out of RAM causing the Java garbage collector to thrash, and Con_BDD times out. The performance gap between concrete and abstract analysis is not constant but increases with network size. Interestingly, there appears to be little additional performance cost to keeping more information (*e.g.*, the destination/origin) in the reachability abstraction (bottom 3 lines). Finally, computing all pairs, all packets dataplane reachabililty is cheap (but not free) relative to the cost of the corresponding abstract control plane analysis (Abs_DP vs. Abs_ROC), and still completes orders of magnitude faster than concrete simulation.

Memory consumption is another important measure of analysis performance. The simpler abstract messages mean that their memory usage is lower. BDDs also enable structural sharing of routes, which further allows redundant portions of the routing table to be uniquely shared across devices. As a result, in the right figure, we find that the memory consumption scales nearly linearly with network size. In contrast, concrete analyses for both Batfish and BDDs scale highly non-linearly. Between these two concrete methods BDDs use less memory than the traditional data structures of Batfish though their analysis time is longer.

8.3 Precision and Speedup (Real Networks)

To gauge the precision of the reachability abstraction (ROC), we compute the final "must reach" routes using both our abstract analysis and Batfish on the production networks, and compare the results. We use the output of Batfish as the ground truth and compute precision as the fraction of

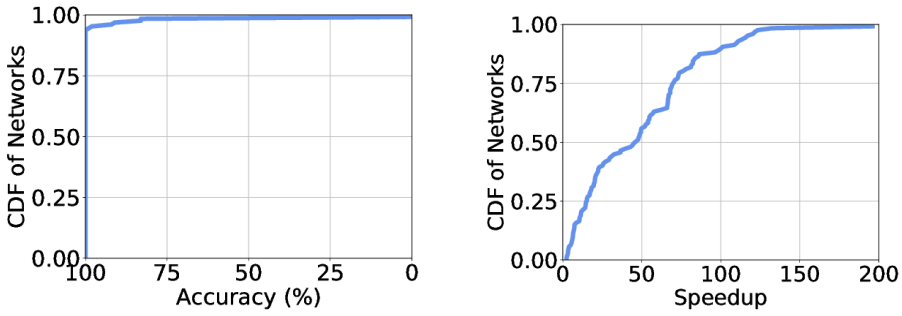

Fig. 9. CDF of Analysis precision (left) and abstraction speedup (right) for production networks.

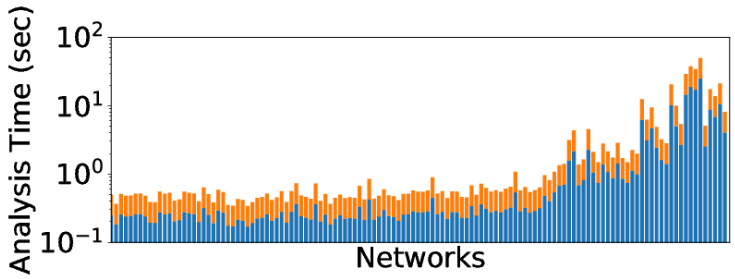

Fig. 10. Abs_ROC (blue) and Abs_DP (orange) performance distributions for production networks. Networks are sorted by size (total configuration size).

Batfish routes that were obtained through the abstract analysis. The left graph of Figure 9 shows a CDF of the abstract analysis precision for the production networks. We find that for about 95% of the production networks, the abstract analysis computes the data plane with 100% accuracy.

Manual inspection of the remaining 5% of the networks for which we lose precision revealed that the main cause is the use of BGP regular expressions, *e.g.*, DC border devices that filter for WAN AS numbers from internal DC routes. Since the abstraction does not track the AS path, it must drop all BGP routes for soundness. Such imprecision could be addressed by keeping additional information in the abstraction, *e.g.*, by tracking if a WAN device has been encountered, or using a regex domain as in Figure 7.

The right graph of Figure 9 shows a CDF of the speedup for the abstract analysis over Batfish for the production networks. For most networks, the speedup ranges between 1 and 2 orders of magnitude. However, we observe that the speedup over Batfish grows rapidly with network size.

Finally, to test the absolute scalability of the abstract analysis, we run the Abs_ROC and Abs_DP analyses on the collection of production networks. Figure 10 shows the analysis time for all networks, where networks are sorted from smallest to largest in terms of total lines of configuration along the x-axis. The largest network is over half a million lines of configuration. The graph shows the time both for control plane reachability (blue) as well as all pairs, all packets dataplane reachability (orange). For all networks studied, the abstract analysis completes in under 40 seconds.

9 RELATED WORK

Abstraction in program analysis. The theory of abstract interpretation [Cousot and Cousot 1977] has been widely used in program verification, with successful applications in industry [Ball et al. 2001; Blanchet et al. 2003]. Different abstract domains, such as intervals [Cousot and Cousot

ARC [Gember-Jacobson et al. 2016]	Path cost, Edge set
BagPipe [Weitz et al. 2016]	Reachability marker, Regex boolean
Batfish [Fogel et al. 2015]	None (sound)
Bonsai [Beckett et al. 2018]	Drop communities
ERA [Fayaz et al. 2016]	Environment (unsound)
Minesweeper [Beckett et al. 2017]	Path length, External communities

Fig. 11. Sources of abstraction in prior work.

1976], octagons [Miné 2001], and polyhedra [Cousot and Halbwachs 1978], provide tradeoffs between precision and scalability. We apply ideas from abstract interpretation to network control planes and formalize a theoretical abstraction framework via routing algebras [Griffin and Sobrinho 2005; Sobrinho 2005]. In some ways, networks are simpler than software programs since there are no loops and thus no need for widening techniques. However, one must still address the challenges of designing and implementing appropriate abstractions based on domain-specific characteristics.

Abstraction in network analysis. The use of abstraction in network analysis is also widespread. Plotkin *et al.* 2016 and Beckett *et al.* 2018 use abstraction to factor out topological symmetries and speed up data plane and control plane analysis, respectively. However, these works focus on *lossless topological* abstractions. We focus on routing message abstraction and provide a continuum of possibly *lossy* abstractions that trade off scalability and precision for *any* network.

Alpernas *et al.* 2018 explored the use of abstract interpretation to analyze isolation properties of stateful data planes. Their stateful models, defined in a relational language called AMDL, represent middleboxes, which include devices such as load balancers, proxies and stateful firewalls. These devices are data plane devices, which forward packets, but record the sets of packets they have seen so far (therein lies the state) and potentially alter their forwarding decisions based on past packets seen. For instance, a stateful firewall may observe a trusted party A sending a message to untrusted party B and henceforth allow B to send messages back to A. Middleboxes usually operate independently and hence such computations are quite different from the distributed control plane protocols that we model. The kinds of abstractions used by Alpernas include (1) abstract away the order in which packets arrive, (2) abstract away the number of packets, (3) abstract away the correlation between states of different middle boxes, and (4) abstract away the correlation between states for different packets within a middlebox. None of these abstractions are similar to our message abstractions and they generally do not make sense in our context.

A number of prior control plane analysis tools use a form of message abstraction (Figure 11). ARC [Gember-Jacobson et al. 2016] abstracts the network by replacing traditional routing messages with a single, global "path cost" value. ARC analyzes graphs that capture useable edges somewhat like "edge set" abstraction of Figure 7 (Row 12). Bagpipe [Weitz et al. 2016] formalizes BGP's semantics and uses an SMT encoding to check control plane properties after abstracting away the underlying IGP and abstracting BGP regex patterns as booleans. Minesweeper [Beckett et al. 2017] also encodes the network using SMT and abstracts the BGP AS path into a cartesian product of several abstractions such as the path length, a loop flag, and more. Additionally, rather than model all community flags that a neighbor can send, Minesweeper abstracts the set of possible external communities into a single abstract value (*i.e.*, "some/none external communities attached"). ERA [Fayaz et al. 2016] and FastPlane [Lopes and Rybalchenko 2019] perform no explicit abstraction but scale by assuming that the network lacks certain complex features. They can be be viewed as using an abstraction that is precise only for such networks and imprecise otherwise. All the tools

above use a fixed abstraction that works well for some networks and fails for others. In contrast, we develop a theory and practical framework that can accommodate a variety of different abstractions.

10 LIMITATIONS AND FUTURE WORK

While this work explores the theory and practical impact of abstractions of routing protocols, there are several limitations and topics worthy of further exploration. For starters, we focus primarily on reachability. While we believe that our methods will be effective for other properties that rely on a single control plane solution (e.g., no private AS number should leak), network properties that require computation of many control plane solutions are typically not addressed effectively via simulation and hence not addressed by our methods. Of course, sometimes, rather than computing many control plane solutions, it may suffice to compute a single, abstract solution. For instance, in §2.4, we illustrated how to summarize many possible inputs to a network with abstract value "E" versus "I." Using that abstraction, we considered many possible hijacking attacks simultaneously. Some properties may resist summarization of this form. For instance, to analyze fault tolerance properties of a network, one might wish to consider the consequences of every possible link failure. While our tool could simulate the consequences of any single link failure (simply by disabling the corresponding edge in the network and running a simulation as is), we do not currently know how to simulate all possible faults simultaneously. Tools such as Minesweeper [Beckett et al. 2017] and Origami [Giannarakis et al. 2019] consider all possible faults at once by encoding networks and their desirable properties as SMT formulae, but they do not scale nearly as well as ShapeShifter. We hope that the kinds of abstractions developed in this work could help scale such other systems in the future.

Recently, researchers have proposed data plane analyses that attempt to check quantitative properties such as those pertaining to quality of service [Foster et al. 2016]. To our knowledge, no one has yet proposed effective algorithms for analyzing quantitative properties of all data planes possibly produced by a control plane. This open problem is bound to be expensive and is well beyond the scope of the current paper. We speculate that abstractions will be useful but they will likely come in quite a different form from the ones we consider as they would have to apply to numerical domains that do not arise in our context.

Another avenue of future research is to address the failure of an abstraction to compute a sufficiently precise answer to a network query. At the moment, a ShapeShifter user must simply choose a more precise abstraction and rerun the analysis. Automated support for doing so will be useful. Our experiments show that even when ShapeShifter fails to analyze a network perfectly, it may analyze many individual destinations precisely. In this case, a good portion of the problem has been solved, leaving a smaller problem with fewer destinations to be solved with a more precise analysis. One can imagine automatically excluding components of the configurations that have already been verified. More interestingly, one can also imagine using counterexample-guided abstraction refinement to automatically choose new abstractions for subsequent verification passes.

11 CONCLUSION

We provide the first formal and systematic account of the role of sound abstraction (with one-sided error) in the analysis of network control planes. We define sufficient conditions for the construction of new sound analyses, and we explore the resulting tradeoff between performance and precision. The notion of abstraction is general, and we show that it is capable of capturing many ideas from the existing verification literature as well as enabling new kinds of analyses. Guided by these ideas, we design a new reachability analysis tool called ShapeShifter that can compute all reachable packets between all pairs of devices, orders of magnitude faster than existing network simulators.

A SOUNDNESS PROOFS

Theorem 4.1. *Consider algebra A with semantics σ and abstract algebra $A^\sharp$ with semantics $\sigma^\sharp$, where A and $A^\sharp$ are related by sound abstraction $(\alpha, \sqsubseteq^\sharp)$. For all times t, concrete instantiations $\pi = (V, E, I, X, \tau, \omega)$, and abstract instantiation $\pi^\sharp = (V, E, \alpha(I), \alpha(X), \tau, \omega)$ we have $\alpha(\sigma^t(\pi)) \sqsubseteq^\sharp \sigma^{\sharp t}(\pi^\sharp)$.*

Proof. By strong induction on the time t.
case $(t = 0)$**:** Evaluate $\alpha(\sigma^0(\pi)) = \alpha(X)$ and also $\sigma^{\sharp 0}(\pi^\sharp) = \alpha(X)$, and $\alpha(X) \sqsubseteq^\sharp \alpha(X)$ since $\sqsubseteq^\sharp$ is reflexive.
case t**:** To show $\alpha(\sigma^t(\pi)) \sqsubseteq^\sharp \sigma^{\sharp t}(\pi^\sharp)$, we must show $\alpha(\sigma^t(\pi))_i \sqsubseteq \sigma^{\sharp t}(\pi^\sharp)_i$ for any i, since these are routing states (vectors). There are two cases to consider.
(1) The first is that $i \notin \tau(t)$. In this case: $\alpha(\sigma^t(\pi))_i = \alpha(\sigma^t(\pi)_i) = \alpha(\sigma^{t-1}(\pi)_i) = \alpha(\sigma^{t-1}(\pi))_i$. From the inductive hypothesis, we know $\alpha(\sigma^{t-1}(\pi))_i \sqsubseteq \sigma^{\sharp t-1}(\pi^\sharp)_i$ which is exactly what we obtain from evaluating $\sigma^{\sharp t}(\pi^\sharp)_i$ under the same schedule.
(2) The second case is where $i \in \tau(t)$. In this case, we have

$$\alpha(\sigma^t(\pi)_i) = \alpha\big(\big(\bigoplus_{(i,j)\in E} f_{ij}(\sigma^{\omega(t,i,j)}(\pi)_j)\big) \oplus I_i\big)$$

and we want to show it is less than $(\sqsubseteq^\sharp)$:

$$\sigma^{\sharp t}(\pi^\sharp)_i = \big(\bigoplus^\sharp_{(i,j)\in E} f^\sharp_{ij}(\sigma^{\sharp\omega(t,i,j)}(\pi^\sharp)_j)\big) \ \oplus^\sharp \ \alpha(I)_i$$

We repeatedly apply the soundness condition: $\alpha(a \oplus b) \sqsubseteq^\sharp \alpha(a) \oplus^\sharp \alpha(b)$ together with commutativity and associativity of $\oplus$ and $\oplus^\sharp$, to show a chain of inequalities. Consider:

$$\alpha\big(\big(\bigoplus_{(i,j)\in E} f_{ij}(\sigma^{\omega(t,i,j)}(\pi)_j)\big) \oplus I_i\big)$$

By repeatedly applying soundness of $\oplus$ for each neighbor j, we get that this is less than $(\sqsubseteq^\sharp)$:

$$\bigoplus^\sharp_{(i,j)\in E} \alpha(f_{ij}(\sigma^{\omega(t,i,j)}(\pi)_j)) \ \oplus^\sharp \ \alpha(I_i)$$

Since $\alpha(I_i)$ is $\alpha(I)_i$ by definition, this is just:

$$\bigoplus^\sharp_{(i,j)\in E} \alpha(f_{ij}(\sigma^{\omega(t,i,j)}(\pi)_j)) \ \oplus^\sharp \ \alpha(I)_i$$

Again, we want to show this is less than $(\sqsubseteq)$

$$\sigma^{\sharp t}(\pi^\sharp)_i = \big(\bigoplus^\sharp_{(i,j)\in E} f^\sharp_{ij}(\sigma^{\sharp\omega(t,i,j)}(\pi^\sharp)_j)\big) \ \oplus^\sharp \ \alpha(I)_i$$

Because $\oplus^\sharp$ is monotone for abstract algebra $A^\sharp$, it suffices to show that each element is smaller pointwise. We easily observe that $\alpha(I)_i \sqsubseteq^\sharp \alpha(I)_i$, so we need to show:

$$\bigoplus^\sharp_{(i,j)\in E} \alpha(f_{ij}(\sigma^{\omega(t,i,j)}(\pi)_j)) \sqsubseteq^\sharp \big(\bigoplus^\sharp_{(i,j)\in E} f^\sharp_{ij}(\sigma^{\sharp\omega(t,i,j)}(\pi^\sharp)_j)\big)$$

From the soundness of F and the monotonicity of $\oplus^\sharp$ we have:

$$\bigoplus^\sharp_{(i,j)\in E} \alpha(f_{ij}(\sigma^{\omega(t,i,j)}(\pi)_j)) \sqsubseteq^\sharp \bigoplus^\sharp_{(i,j)\in E} f^\sharp_{ij}(\alpha(\sigma^{\omega(t,i,j)}(\pi^\sharp)_j))$$

And now we must prove that:

$$\bigoplus^\sharp_{(i,j)\in E} f^\sharp_{ij}(\alpha(\sigma^{\omega(t,i,j)}(\pi^\sharp)_j)) \sqsubseteq^\sharp \bigoplus^\sharp_{(i,j)\in E} f^\sharp_{ij}(\sigma^{\sharp\omega(t,i,j)}(\pi^\sharp)_j)$$

From the monotonicity of $\oplus^\sharp$, we can show the inequality pointwise:

$$f_{ij}^\sharp(\alpha(\sigma^{\omega(t,i,j)}(\pi)_j)) \sqsubseteq^\sharp f_{ij}^\sharp(\sigma^{\sharp\omega(t,i,j)}(\pi^\sharp)_j)$$

From the monotonicity of $F^\sharp$, we can simply show the values inside the transfer function respect $\sqsubseteq^\sharp$:

$$\alpha(\sigma^{\omega(t,i,j)}(\pi)_j) \sqsubseteq^\sharp \sigma^{\sharp\omega(t,i,j)}(\pi^\sharp)_j$$

From the definition of α on routing states, this is the same as showing:

$$\alpha(\sigma^{\omega(t,i,j)}(\pi))_j \sqsubseteq^\sharp \sigma^{\sharp\omega(t,i,j)}(\pi^\sharp)_j$$

Finally, we observe that this is exactly the inductive hypothesis, which we can apply in this case since we have $\omega(t,i,j) < t$ by definition because messages take non-zero time to be transmitted. □

Theorem 4.2. *Consider algebra A with semantics σ, and abstract algebra $A^\sharp$ with semantics $\sigma^\sharp$, related by sound abstraction $(\alpha, \sqsubseteq^\sharp)$. If $A^\sharp$ is uniquely converging for topology (V,E) and states $(\alpha(I), \alpha(X))$, then for all schedules (τ_1,ω_1) and (τ_2,ω_2), there exists a time t such that for all k, $\alpha(\sigma^{t+k}(\pi)) \sqsubseteq^\sharp \sigma^{\sharp t+k}(\pi^\sharp)$ for $\pi = (V,E,I,X,\tau_1,\omega_1)$, and $\pi^\sharp = (V,E,\alpha(I),\alpha(X),\tau_2,\omega_2)$.*

Proof. Since the abstract routing algebra converges uniquely, we know that at some time t, the final state (X^*) will be the same regardless of schedule. Thus there is a t such that for all k:

$$\sigma^{\sharp t+k}(\pi^\sharp) = \sigma^{\sharp t+k}(V,E,\alpha(I),\alpha(X),\tau_1,\omega_1)$$

We then appeal to Theorem 4.1 since to give us the result:

$$\alpha(\sigma^{t+k}(V,E,I,X,\tau_1,\omega_1)) \sqsubseteq^\sharp \sigma^{\sharp t+k}(V,E,\alpha(I),\alpha(X),\tau_1,\omega_1)$$

for any such k. It follows from equality that:

$$\alpha(\sigma^{t+k}(V,E,I,X,\tau_1,\omega_1)) \sqsubseteq^\sharp \sigma^{\sharp t+k}(V,E,\alpha(I),\alpha(X),\tau_2,\omega_2)$$

where (τ_1, ω_1) and (τ_2, ω_2) may be different schedules. □

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation awards NeTS 1704336 and FMitF 1837030, DARPA Dispersed Computing program under award number HR0011-17-C-0047 and a Facebook Research Award on "Network control plane verification at scale."

REFERENCES

Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM*.

Kalev Alpernas, Roman Manevich, Aurojit Panda, Mooly Sagiv, Scott Shenker, Sharon Shoham, and Yaron Velner. 2018. Abstract Interpretation of Stateful Networks. In *Static Analysis Symposium*.

Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. 2014. NetKAT: Semantic Foundations for Networks. In *POPL*.

Thomas Ball, Rupak Majumdar, Todd D. Millstein, and Sriram K. Rajamani. 2001. Automatic Predicate Abstraction of C Programs. In *PLDI*. 203–213.

Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A General Approach to Network Configuration Verification. In *SIGCOMM*.

Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2018. Control Plane Compression. In *SIGCOMM*. 476–489.

Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. 2003. A static analyzer for large safety-critical software. In *PLDI*. 196–207.

Randal E. Bryant. 1986. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Computers* 35, 8 (1986), 677–691.

Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2000. Counterexample-Guided Abstraction Refinement. In *Computer Aided Verification, 12th International Conference, CAV, Proceedings*. 154–169.

Clos Network 2019. Clos network. https://en.wikipedia.org/wiki/Clos$_n$etwork.

Patrick Cousot and Radhia Cousot. 1976. Static determination of dynamic properties of programs. In *Proceedings of the 2nd International Symposium on Programming, Paris, France*. Dunod, 106–130.

Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages*. 238–252.

Patrick Cousot and Nicolas Halbwachs. 1978. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL*. 84–96.

Matthew L. Daggitt, Alexander J. T. Gurney, and Timothy G. Griffin. 2018a. Asynchronous Convergence of Policy-rich Distributed Bellman-ford Routing Protocols. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 103–116.

Matthew L. Daggitt, Alexander J. T. Gurney, and Timothy G. Griffin. 2018b. Asynchronous Convergence of Policy-rich Distributed Bellman-ford Routing Protocols. In *SIGCOMM*. 103–116.

Xenofontas A. Dimitropoulos and George F. Riley. 2006. Efficient Large-scale BGP Simulations. *Comput. Netw.* 50, 12 (August 2006), 2013–2027.

A. Fabrikant, U. Syed, and J. Rexford. 2011. There's something about MRAI: Timing diversity can exponentially worsen BGP convergence. In *INFOCOM*. 2975–2983.

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 *OSDI*.

Nick Feamster. 2005. *Proactive Techniques for Correct and Predictable Internet Routing*. Ph.D. Dissertation. Massachusetts Institute of Technology.

Gilberto Filé and Francesco Ranzato. 1999. The powerset operator on abstract interpretations. *Theoretical Computer Science* 222, 1 (1999), 77 – 111.

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 *NSDI*.

Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. 2016. Probabilistic NetKAT. In *Proceedings of the 25th European Symposium on Programming Languages and Systems - Volume 9632*. 282–309.

Lixin Gao and Jennifer Rexford. 2000. Stable Internet Routing Without Global Coordination. In *SIGMETRICS*.

Aaron Gember-Jacobson, Raajay Viswanathan, Aditya Akella, and Ratul Mahajan. 2016. Fast Control Plane Analysis Using an Abstract Representation. In *SIGCOMM*.

Nick Giannarakis, Ryan Beckett, Ratul Mahajan, and David Walker. 2019. *Efficient Verification of Network Fault Tolerance via Counterexample-Guided Refinement*. 305–323.

Timothy G. Griffin, F. Bruce Shepherd, and Gordon Wilfong. 2002. The Stable Paths Problem and Interdomain Routing. *IEEE/ACM Trans. Networking* 10, 2 (2002).

Timothy G. Griffin and Joäo Luís Sobrinho. 2005. Metarouting. In *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 1–12.

Peyman Kazemian, Michael Chang, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. 2013. Real Time Network Policy Checking Using Header Space Analysis. In *NSDI*. 99–112.

Peyman Kazemian, George Varghese, and Nick McKeown. 2012. Header Space Analysis: Static Checking for Networks. In *NSDI*.

Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. 2013. VeriFlow: Verifying Network-Wide Invariants in Real Time. In *NSDI*.

Matthew L. Daggitt and Timothy Griffin. 2018. Rate of Convergence of Increasing Path-Vector Routing Protocols. In *ICNP*. 335–345.

P. Lapukhov, A. Premji, and J. Mitchell. 2015. Use of BGP for routing in large-scale data centers. Internet draft.

Olle Liljenzin. 2013. Confluently Persistent Sets and Maps. *CoRR* abs/1301.3388 (2013).

Nuno Lopes, Nikolaj Bjorner, Patrice Godefroid, Karthick Jayaraman, and George Varghese. 2015. Checking Beliefs in Dynamic Networks. In *NSDI*.

Nuno P. Lopes and Andrey Rybalchenko. 2019. Fast BGP Simulation of Large Datacenters. In *Proc. of the 20th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*.

Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. 2011. Debugging the Data Plane with Anteater. In *SIGCOMM*.

Antoine Miné. 2001. The Octagon Abstract Domain. In *Proceedings of the Eighth Working Conference on Reverse Engineering, (WCRE)*. 310–319.

Raphaël Monat and Antoine Miné. 2017. Precise Thread-Modular Abstract Interpretation of Concurrent Programs Using Relational Interference Abstractions. In *Verification, Model Checking, and Abstract Interpretation*, Ahmed Bouajjani and David Monniaux (Eds.). Springer International Publishing, 386–404.

Sanjay Narain, Dana Chee, Brian Coan, Ben Falchuk, Samuel Gordon, Jaewon Kang, Jonathan Kirsch, Aditya Naidu, Kaustubh Sinkar, and Simon Tsang. 2016. A Science of Network Configuration. *Journal of Cyber Security and Information Systems* 1, 4 (2016).

Gordon D. Plotkin, Nikolaj Bjørner, Nuno P. Lopes, Andrey Rybalchenko, and George Varghese. 2016. Scaling Network Verification Using Symmetry and Surgery. In *POPL*.

Santhosh Prabhu, Ali Kheradmand, Brighten Godfrey, and Matthew Caesar. 2017. Predicting Network Futures with Plankton. In *Proceedings of the First Asia-Pacific Workshop on Networking (APNet'17)*. 92–98.

Redistributing Routing Protocols 2012. Redistributing Routing Protocols. https://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/8606-redist.html.

D Roberts. 2018. It's been a week and customers are still mad at BB&T. https://www.charlotteobserver.com/news/business/banking/article202616124.html.

Simon Sharwood. 2016. Google cloud wobbles as workers patch wrong routers. http://www.theregister.co.uk/2016/03/01/google$_{c}$loud$_{w}$obbles$_{a}$s$_{w}$orkers$_{p}$atc$_{w}$rong$_{r}$outers/.

João Luís Sobrinho. 2005. An Algebraic Theory of Dynamic Network Routing. *IEEE/ACM Trans. Netw.* 13, 5 (October 2005), 1160–1173.

Yevgenly Sverdlik. 2012. Microsoft: misconfigured network device led to Azure outage. http://www.datacenterdynamics.com/content-tracks/servers-storage/microsoft-misconfigured-network-device-led-to-azure-outage/68312.fullarticle.

Y Sverdlik. 2017. 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.

Dylan Tweney. 2013. 5-minute outage costs Google $545,000 in revenue. https://venturebeat.com/2013/08/16/3-minute-outage-costs-google-545000-in-revenue/.

Kannan Varadhan, Ramesh Govindan, and Deborah Estrin. 1996. *Persistent route oscillations in inter-domain routing*. Technical Report. Computer Networks.

Anduo Wang, Limin Jia, Wenchao Zhou, Yiqing Ren, Boon Thau Loo, Jennifer Rexford, Vivek Nigam, Andre Scedrov, and Carolyn L. Talcott. 2012. FSR: Formal Analysis and Implementation Toolkit for Safe Inter-domain Routing. *IEEE/ACM Trans. Networking* 20, 6 (2012).

Konstantin Weitz, Doug Woos, Emina Torlak, Michael D. Ernst, Arvind Krishnamurthy, and Zachary Tatlock. 2016. Formal Semantics and Automated Verification for the Border Gateway Protocol. In *NetPL*.