Control Plane Compression

Ryan Beckett
Princeton University

Aarti Gupta
Princeton University

Ratul Mahajan
Intentionet

David Walker
Princeton University

ABSTRACT

We develop an algorithm capable of compressing large networks into smaller ones with similar control plane behavior: For every stable routing solution in the large, original network, there exists a corresponding solution in the compressed network, and vice versa. Our compression algorithm preserves a wide variety of network properties including reachability, loop freedom, and path length. Consequently, operators may speed up network analysis, based on simulation, emulation, or verification, by analyzing only the compressed network. Our approach is based on a new theory of control plane equivalence. We implement these ideas in a tool called Bonsai and apply it to real and synthetic networks. Bonsai can shrink real networks by over a factor of 5 and speed up analysis by several orders of magnitude.

CCS CONCEPTS

• **Networks** → **Network reliability**; **Network management**; • **Software and its engineering** → *Formal methods*;

KEYWORDS

Network Verification, Stable Routing Problem

ACM Reference Format:
Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2018. Control Plane Compression. In *SIGCOMM '18: ACM SIGCOMM 2018 Conference, August 20–25, 2018, Budapest, Hungary*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3230543.3230583

1 INTRODUCTION

Configuration errors are a leading cause of network outages and security breaches [3, 21, 28, 33, 35, 40]. For instance, a recent misconfiguration disrupted Internet connectivity for millions of users in the USA for over 1.5 hours, and similar incidents last year impacted users in Japan, India, Brazil, Azerbaijan, and beyond [7].

The root cause of many of these errors is simply complexity: Networks typically run one or more distributed routing protocols that exchange information about available paths to destinations. The paths that are advertised and preferred by routers are determined by their configuration files, which can easily contain tens of thousands of low-level, vendor-specific primitives. Unfortunately, it is nearly impossible for human network operators to reason about the correctness of these files, let alone that of behavior that results from distributed interactions between many routers.

To minimize errors in configurations, researchers have taken a number of approaches to find bugs and check correctness, including static analysis [5, 16], simulation [17, 19], emulation [31], monitoring [25, 26, 29, 44], model checking [37], systematic testing [15, 39], and verification [6, 20, 42]. Yet for almost all such techniques, scaling to large networks remains challenging. For example, in Batfish [19], a testing tool, the time it takes to model control plane dynamics limits the number of tests that can be administered. Similarly, the cost of the verification tool Minesweeper [6] grows exponentially in the worst case, and in practice, tops out at a few hundred devices—far short of the 1000+ devices that are used to operate many modern data centers.

In this paper, we tackle these problems by defining a new theory of *control plane equivalence* and using it to compress large, concrete networks into smaller, abstract networks with equivalent behavior. Because our compression techniques preserve many properties of the network control plane, including reachability, path length, and loop freedom, analysis tools of all kinds can operate (quickly) on the smaller networks, rather than their large concrete counterparts. In other words, this theory is an effective complement to ongoing work on network analysis, capable of helping accelerate a wide variety of analysis tools. Moreover, because our transformations are bisimulations, rather than over- or under-approximations, tools built on our theory can avoid both unsound inferences and false positives.

Intuitively, the reason it is possible to compress control planes in this fashion is that large networks tend to contain quite a bit of structural symmetry—if not, they would be even harder to manage. For instance, many spine (or leaf or aggregation) routers in a data center may be similarly configured; and, as we show later, symmetries exist in backbone network as well. Recently, Plotkin *et al.* [36] exploited similar intuition to develop other tools for network verification. However, they operate over the (stateless) network data plane, *i.e.,* the packet-forwarding rules, whereas we operate over the control plane, *i.e.,* the protocols that distribute the available routes. While both the data and control planes process messages (data packets and routing messages, respectively), the routing messages interact with one another whereas the data packets do not. More specifically, data packet processing depends only on the static packet-forwarding rules of a router; it does not depend on other data packets. In contrast, routing messages interact: the presence and timing of one (more preferred) message can cause another (less preferred) message to be ignored. Such interactions create dynamics not present in stateless data planes and can even lead to many different routing solutions for the same network. In other work, Wang *et al.* [41] also observed that compression can lead to improved control plane analysis performance, and they designed an algorithm for compressing BGP networks that can speed analysis of convergence behavior. In contrast, our algorithms are designed to preserve arbitrary path properties of networks. Such properties include reachability, loop freedom, absence of black holes and access control. We also define our algorithms over a generic protocol model so we can apply the ideas to a wide range of protocols ranging from BGP to OSPF to static routes.

More specifically, we make two core contributions:

A Theory of Control Plane Equivalence. Our theoretical development begins by defining the Stable Routing Problem (SRP), a generic model of a routing protocol and the network on which it runs. SRPs can model networks running a wide variety of protocols including distance-vector, link-state, and path-vector protocols. SRPs are directly inspired by the stable paths problems (SPP) [22], but rather than describing the protocols' final solution using end-to-end paths (as SPPs do), SRPs describe runtime routing behavior in terms of local processing of routing messages, as configurations do. In addition to modeling raw configurations more closely, this distinction allows SRPs to capture a wider variety of routing behaviors that emerge at runtime, including static routing and other protocols that may generate loops. Consequently, our formulation of SRPs is similar to the protocol models used by routing algebras [23, 38], though this earlier work focused on convergence rather than network compression or analysis of topologically-sensitive properties such as reachability.

With a network model in hand, we turn to the process of characterizing network transformations. Intuitively, we would like to define transformations that convert concrete networks into abstract ones that make equivalent control decisions and generate similar forwarding behavior. However, doing so directly is challenging as validating that two SRPs are equivalent is as hard as the control plane verification problem we are trying to speed up in the first place! We address this challenge by defining a class of network transformations that we call *effective abstractions*. These abstractions are characterized by conditions designed to be checked efficiently, and locally, without the need for a global simulation. Our central theoretical result is that these conditions imply behavioral equivalence of the concrete and abstract networks.

An Efficient Compression Algorithm. Our theory paves the way for a practical algorithm for automatically computing compact, abstract control planes from configurations of large networks. The algorithm is based on abstraction refinement: Starting with the coarsest possible abstraction it iteratively refines the abstract network, checking at each step to determine whether or not it has found an effective abstraction. To implement such checks efficiently, we use a variety of efficient data structures such as Binary Decision Diagrams to represent configuration semantics. In practice, the algorithm reduces network sizes significantly, bringing more networks into range for various analyses. For example, our tool was able to compress an operational 196-node data center network down to 26 nodes and to reduce the number of edges by a factor of roughly 100. A 1086-node WAN using eBGP, iBGP, OSPF and static routes was compressed down to 137 nodes and its edge count was reduced by a factor 7.

2 OVERVIEW

The Stable Routing Problem. Intuitively, a network is just a graph G where nodes are routers and edges are links between them. The network's control plane has a collection of router-local rules that determine how routing messages are processed. Different routing protocols use different kinds of messages. For instance, in RIP, a simple distance-vector protocol, messages include destination prefix and hop count. In contrast, BGP messages include a destination prefix, an AS-path, a local preference and other optional data. We call all such messages *attributes* regardless of their contents. While routing protocols differ significantly in many respects, they can be factored into two generic parts: (1) a comparison relation that prefers certain attributes, and (2) a transfer function that transforms incoming and outgoing messages.

An SRP instance assembles all of these ingredients: (1) a graph defining the network topology, (2) a destination to which to route, (3) a set of attributes, (4) an attribute comparison relation, and (5) an attribute transfer function. Its *solution* ($\mathcal{L}$) associates an attribute with each node, which represents the route chosen by the node. Every SRP solution has the property that nodes have not been offered an attribute by a neighbor that is preferred more than the chosen one. An SRP solution also implicitly defines a forwarding relation: If a receives its chosen attribute from b, then a will forward traffic to b. There can be multiple solutions to an SRP.

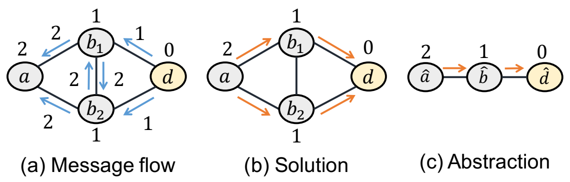


Figure 1: An example RIP network.

As an example, consider the RIP network in Figure 1(a). The destination node is d. It initiates the protocol by sending messages that contain the hop count to the destination. The RIP comparison relation prefers the minimal attribute (i.e., the shortest path to the destination). The RIP transformation function adds one to each attribute along each link. Figure 1(b) shows the resulting solution.

Network Abstractions. Our goal is to define an algorithm that, given one SRP, computes a new, smaller SRP that exhibits "similar" control plane behavior. We call the input SRP the *concrete network*, and the output SRP (denoted $\widehat{SRP}$) the *abstract network*. A *network abstraction* defines precisely the relationship between the two. It is a pair of functions (f, h), where f is a *topology abstraction* that maps the nodes and edges of the concrete network to those of the abstract network, and h is an *attribute abstraction* that maps the concrete attributes in control plane messages to abstract ones.

We define "similarity" using control plane solutions that emerge after running a routing protocol. More precisely, two networks are *control-plane equivalent (CP-equivalent)* when:

> *There is a solution $\mathcal{L}$ to the concrete network iff there is a solution $\widehat{\mathcal{L}}$ to the abstract network where (i) routers are labeled with similar attributes, as defined by the attribute abstraction; and (ii) packets are forwarded similarly, as defined by the topology abstraction.*

CP-equivalence is powerful because it preserves many properties such as reachability, path length, way-pointing, and loop-freedom. Moreover, because the connection between abstract and concrete networks is tight (*i.e.*, a bisimulation) as opposed to an over-approximation, bugs found when verifying the abstract network, correspond to real bugs in the concrete network (*i.e.*, no false positives). Likewise, because our abstractions are not under-approximations, if we verify that there are no violations of a property in the abstract network, then there are no violations of the property in the concrete network (*i.e.*, no false negatives).

Figure 1(c) shows a CP-equivalent abstraction of the example network. The topology abstraction f maps the concrete node a to $\widehat{a}$, b_1 and b_2 to $\widehat{b}$, and d to $\widehat{d}$, while the attribute abstraction h is simply the identity function, leaving hop count unchanged. The abstraction is CP-equivalent because there is only one stable solution to both abstract and concrete networks, and given a concrete node n, the label associated with that node is the same as the label associated with $f(n)$. For instance, b_1 is labeled with attribute 1 and so is $\widehat{b}$, its corresponding node in the abstraction. One can also observe that the forwarding relation in the concrete network is equivalent (modulo f) to the forwarding relation in the abstract network. For instance, concrete node b_1 forwards to d and the corresponding abstract node $\widehat{b}$ forwards to $\widehat{d}$ as well.

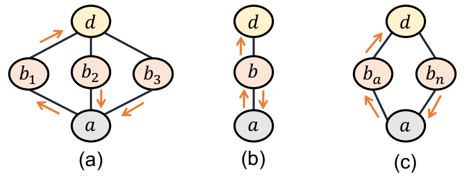


Figure 2: (a) Concrete BGP network. (b) Unsound abstraction (has a loop). (c) Sound abstraction.

Effective Abstractions. While CP-equivalence is our goal, we cannot evaluate pairs of networks for equivalence directly—naively, one would have to simulate the behavior of the pair of networks on all possible inputs, an infeasible task. Instead, we formulate a set of conditions on network abstractions that imply CP-equivalence and can be evaluated efficiently. *Effective abstractions* are those that satisfy these conditions.

While these conditions help us identify abstractions for protocols such as RIP and OSPF, there is a serious complication for BGP. One of the conditions is *transfer-equivalence*, *i.e.,* the routing information is transformed in a similar way in concrete and abstract networks. However, BGP routers employ an implicit loop-prevention mechanism that rejects routes that contain their own AS (Autonomous System, an identifier for the network) number. Consequently, even when two routers have identical configurations, their transfer functions are slightly different because they reject different paths.

To handle this complication, we define a refined set of conditions, called the *BGP-effective conditions.* These conditions also imply CP-equivalence and can be evaluated efficiently, though the relationship between abstract and concrete networks is more sophisticated; the function mapping nodes in the concrete to the abstract networks is not fixed but instead depends on the (one of possibly multiple) solutions to which the control plane converges.

More precisely, given a concrete SRP and an effective abstraction, which produces $\widehat{SRP}$, a BGP-effective abstraction provides an intermediate network $\overline{SRP}$. This intermediate network is similar to $\widehat{SRP}$ except that an abstract node $\widehat{n}$ in $\widehat{SRP}$ is split into several nodes—one for each possible forwarding behavior of $\widehat{n}$. Importantly, we prove that the number of instances node $\widehat{n}$ needs to be split into, is bounded by k, where k is the number of different local preference values that the concrete nodes may use. (Operators use local preferences to implement policy-based path selection.)

Figure 2 shows a situation in which these sorts of difficulties arise. Assume the middle routers (b_1, b_2, b_3) of the concrete network have identical configurations and prefer to route traffic down rather than up. Despite this preference, one of the three must route upwards. In the figure, b_1 happens to be that router. This solution is stable—no router receives a route from a neighbor that it prefers to the current route (if router b_1 were to receive a route from a, the path to d would be $b_1.a.b_1.d$, a loop which b_1 would reject). And yet, despite identical configurations, routers b_1 and b_2 forward in different directions. Figure 2(b) shows a naive (and incorrect) abstraction in which all three of b_1, b_2 and b_3 are collapsed to the same node. This abstract network in (b) is not CP-equivalent to the network in (a), because mapping the solution to (a) in (b) requires generating a forwarding loop. However, there does exist a smaller CP-equivalent abstract network—the network depicted in Figure 2(c). The latter network is capable of mapping the solution depicted in Figure 2(a) without introducing a forwarding loop.

From Theory to Practice. Our theory provides the basis for developing an efficient algorithm for control plane compression. Based on *abstraction refinement*, our algorithm first generates the coarsest possible abstraction and then repeatedly splits abstract nodes until the resulting network satisfies the conditions of an (BGP-)effective abstraction.

Figure 3 visualizes the algorithm on the BGP network of Figure 2(a). As a first step in Figure 3(a), we generate the coarsest possible abstraction: the destination is represented alone as one abstract node and all other nodes are grouped in a separate abstract node. This first abstraction is not an effective abstraction—it does not satisfy a topological condition requiring that all concrete nodes (b_1, b_2, b_3, a) associated with one abstract node have edges to some concrete node (d) in an adjacent abstract node. In this case, concrete node a does not satisfy the condition. It is thus necessary to refine the abstraction by separating nodes b_1, b_2, and b_3 from a.

Figure 3(b) presents the second refinement step, where the topological condition is satisfied but the BGP-effective conditions are not: The nodes b_1, b_2, and b_3 use one non-default BGP local preference to prefer routing down rather than up and as a consequence each node may exhibit up to two possible behaviors. Consequently, we must split the abstract node for b_1, b_2, and b_3 into two separate nodes. We do not know statically the mapping of concrete to abstract nodes, so our visualization places all three concrete nodes in each abstract node to represent all possible mappings.

Figure 3(c) happens to satisfy all conditions of a BGP-effective abstraction. Consequently, the refinement process terminates. The final abstraction includes 4 abstract nodes and 4 total edges—a reduction in size from our concrete network with 5 nodes and 6 edges. Although this simple

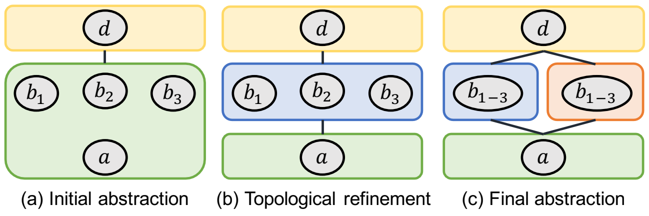


Figure 3: Abstraction refinement for the network in Figure 2(a). Boxes represent abstract nodes.

example does not show much reduction, as we show later, significant reductions are possible in larger networks.

Onward. The following sections describe our approach in detail. §3 formalizes the SRP, §4 defines effective abstractions, and §5 describes the compression algorithm. Throughout the paper there are many theorems. The proofs of these theorems can be found in the accompanying technical report [1].

3 STABLE ROUTING PROBLEM

An SRP formally captures all the routing behaviors that a network can exhibit. We first define it formally and then outline how it can model common routing protocols.

3.1 Definition and Solutions

We define one SRP per destination in the network. As shown in Figure 4, an SRP instance is a tuple $(G, A, a_d, \prec, \text{trans})$. Here, $G = (V, E, d)$ is a graph with a set of vertices V, a set of directed edges $E : V \times V$, and a destination vertex $d \in V$. A is a set of *attributes* that describe the fields of routing messages. For example, when modeling BGP, A might represent tuples of a 32-bit local-preference value, a set of 16-bit community values, and a list of ASes representing the AS path. We also define a new set $A_\perp = A \cup \{\perp\}$, which adds a special value $\perp$ to represent the absence of an attribute (routing message). Further, the special attribute value a_d represents the initial protocol message advertised by the destination d.

In the SRP instance, $\prec$ is a partial order that compares attributes and models the routing decision procedure that compares routes using some combination of message fields. If attribute $a_1 \prec a_2$, then a_1 is more desirable. Finally, trans represents the transfer function that describes how attributes are modified (or dropped) between routers. Given an edge and an attribute from the neighbor across the edge, it determines what new attribute is received at the current node. The transfer function depends on both the routing protocol and node's configuration.

Well-formed SRPs. In an SRP, the $\prec$ relation and trans function can compare and modify attributes arbitrarily. While this generality helps model a wide variety of routing protocols, it also allows nonsensical behaviors. We define *well-formed*

SRP instance $\boxed{SRP = (G, A, a_{\text{d}}, \prec, \text{trans})}$

G	$=$	(V, E, d)	*network topology*
V			*topology vertices*
E	$:$	$V \times V$	*topology edges*
d	$:$	V	*destination vertex*
A			*routing attributes*
$A_\bot$	$=$	$A \cup \{\bot\}$	*attributes or no attribute*
a_{d}	$:$	$A_\bot$	*initial route*
$\prec$	$\subseteq$	$A \times A$	*comparison relation*
trans	$:$	$E \times A_\bot \to A_\bot$	*transfer function*

Properties of well-formed SRPs

$\forall v.\ (v, v) \notin E$	*self-loop-free*
$\forall e.\ \text{trans}(e, \bot) = \bot$	*non-spontaneous*

SRP solution $\boxed{\mathcal{L} : V \to A}$

$$\mathcal{L}(u) = \begin{cases} a_{\text{d}} & u = d \\ \bot & \text{attrs}_{\mathcal{L}}(u) = \emptyset \\ a \in \text{attrs}_{\mathcal{L}}(u) \text{ that is minimal by } (\prec), & \text{attrs}_{\mathcal{L}}(u) \neq \emptyset \end{cases}$$

$$\text{attrs}_{\mathcal{L}}(u) \;\;= \{a \mid (e, a) \in \text{choices}_{\mathcal{L}}(u)\}$$
$$\text{choices}_{\mathcal{L}}(u) = \{(e, a) \mid e = (u, v),\ a = \text{trans}(e, \mathcal{L}(v)), a \neq \bot\}$$
$$\text{fwd}_{\mathcal{L}}(u) \;\;= \{e \mid (e, a) \in \text{choices}_{\mathcal{L}}(u), a \approx \mathcal{L}(u)\}$$

$$a_1 \approx a_2 \iff a_1 \not\prec a_2 \wedge a_2 \not\prec a_1$$

Network abstraction $\boxed{(f, h) : (V \to \widehat{V}) \times (A \to \widehat{A})}$

$SRP = (G, A, a_{\text{d}}, \prec, \text{trans})$	*concrete SRP instance*
$\widehat{SRP} = (\widehat{G}, \widehat{A}, \widehat{a_{\text{d}}}, \widehat{\prec}, \widehat{\text{trans}})$	*abstract SRP instance*
$u \mapsto \widehat{u} \;\equiv\; f(u) = \widehat{u}$	*vertex abstraction notation*
$a \mapsto \widehat{a} \;\equiv\; h(a) = \widehat{a}$	*attribute abstraction notation*

Effective abstractions

$(d \mapsto \widehat{d}) \wedge (\forall d'.\ d \neq d' \implies d' \not\mapsto \widehat{d})$	*dest-equivalence*
$h(a_d) = \widehat{a_d}$	*orig-equivalence*
$\forall a.\ h(a) = \bot \iff a = \bot$	*drop-equivalence*
$\forall a, b.\ a \prec b \iff h(a) \mathrel{\widehat{\prec}} h(b)$	*rank-equivalence*
$\forall e, a.\ h(\text{trans}(e, a)) = \widehat{\text{trans}}(f(e), h(a))$	*trans-equivalence*
$\forall u, v.\ (u, v) \in E \Rightarrow (\widehat{u}, \widehat{v}) \in \widehat{E}$	∀∃−abstraction1
$\forall \widehat{u}, \widehat{v}.\ (\widehat{u}, \widehat{v}) \in \widehat{E} \Rightarrow (\forall u.\ u \mapsto \widehat{u} \Rightarrow \exists v.\ v \mapsto \widehat{v} \wedge (u, v) \in E)$	∀∃−abstraction2

BGP-effective abstractions

$\forall u, v.\ (u, v) \in E \iff (\widehat{u}, \widehat{v}) \in \widehat{E}$	∀∀−abstraction
$\forall e, a.\ e = (u, v) \ \wedge\ v \notin a.\text{path} \implies h(\text{trans}(e, a)) = \widehat{\text{trans}}(f(e), h(a))$	*transfer-approx*

CP-equivalence $\boxed{SRP \equiv \widehat{SRP}}$

$\mathcal{L} \in SRP \iff \widehat{\mathcal{L}} \in \widehat{SRP}$ when:

1. $\forall u.\ h(\mathcal{L}(u)) = \widehat{\mathcal{L}}(f(u))$	*label-equivalence*
2. $\forall u, v.\ (u, v) \in \text{fwd}_{\mathcal{L}}(u) \iff (\widehat{u}, \widehat{v}) \in \widehat{\text{fwd}}_{\widehat{\mathcal{L}}}(\widehat{u})$	*fwd-equivalence*

Figure 4: Technical cheat sheet. Definitions for SRPs, solutions, abstractions, and abstraction properties.

SRPs as those with two practical properties: (1) *self-loop-freedom*: The graph must not contain self loops: $\forall v.(v, v) \notin E$. (2) *non-spontaneity*: If a neighbor has no route to the destination, then a router will not obtain a route from that neighbor. While useful, non-spontaneity is not necessary for all of our theoretical results (e.g., see SRPs for static routing).

Solutions. Given an SRP instance, we can describe its (possibly multiple) solutions. Intuitively, each solution is derived from a set of constraints that requires that each node be *locally stable*, *i.e.*, it has no incentive to deviate from its currently chosen neighbor. For shortest path routing, an SRP solution will be a rooted tree where each node points to the neighbor with the shortest path. For policy-based routing such as BGP, the paths may not be the shortest paths.

Formally, an SRP solution is an attribute labeling $\mathcal{L} : V \to A$ that maps each node to a final route (attribute) chosen to forward traffic. The labeling $\mathcal{L}$ must satisfy the constraints shown in Figure 4 (lower left). The labeling of the destination node should be the special attribute a_{d}. If there are no attributes available from neighbors ($\text{attrs}_{\mathcal{L}}(u) = \emptyset$), then node u has no route to the destination ($\bot$). Otherwise, $\mathcal{L}(u)$ is chosen to be an attribute choice that is minimal according to the comparison relation ($\prec$). If there is more than one minimal attribute, then any value can be chosen. The set of attributes at a node stems from the choices from neighbors: for each edge $e = (u, v)$ from u, apply the transfer function from the neighbor's label to obtain a new attribute $a = \text{trans}(e, \mathcal{L}(v))$, ignoring any attributes that get dropped ($a = \bot$).

Given an SRP solution, it is easy to determine the forwarding behavior. We define $\text{fwd}_{\mathcal{L}}(u)$ as the set of edges e such that the attribute learned from e is as good as the best choice $\mathcal{L}(u)$ at u. The attribute need not be exactly $\mathcal{L}(u)$, but must be at least as good ($\approx$). If there is more than one such choice, then a node may forward to multiple neighbors.

3.2 Modeling Common Routing Protocols

SRP can faithfully model common routing protocols. For ease of exposition, assume for now that the network runs only one routing protocol; we consider multi-protocol networks and other configuration primitives in §6.

RIP (distance vector). RIP uses shortest paths to the destination based on hop count. The attributes, representing the path length, are $A = \{0..15\}$ as RIP uses a maximum path length of 16; the destination attribute a_{d} is 0; the comparison relation prefers shorter paths; and the transfer function drops a route if it exceeds the hop count limit and increments the path length otherwise.

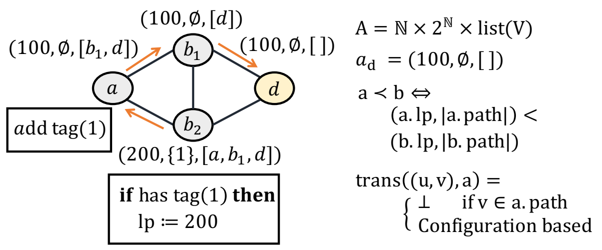


Figure 5: Modeling BGP with SRP.

OSPF (link state). Open Shortest Path First is a popular link state protocol where routers exchange link cost information and compute the least-cost path to the destination. The attribute set $A = \mathbb{N}$ is any natural number and represents paths cost; the comparison relation compares this cost; and the transfer function adds the (configured) link cost. A large OSPF network may be split into multiple areas and prefer intra-area routes over inter-area ones. We model this behavior using attributes that are tuples of the path cost and a boolean that indicates whether it is an inter-area route. The comparison relation prioritizes intra-area routes followed by path cost, and the transfer function changes the boolean value when crossing an inter-area edge.

BGP (path vector). BGP is a widely-used path-vector protocol that provides flexibility for configuring policy and computing non-shortest paths. We assume here that all routers use their own AS number, i.e., eBGP (as in large data centers [30]) and discuss iBGP in §6. We model eBGP using $A = \mathbb{N} \times 2^{\mathbb{N}} \times list(V)$, where the components are: (1) a local preference value, (2) a collection of community tags, and (3) a list of nodes defining the AS path. (Other BGP attributes such as MEDs or origin type can be modeled similarly, but are omitted for simplicity.) BGP's comparison function first compares local-preference followed by the AS path length. Its transfer function appends the current AS to the AS path when exporting a route. It also drops attributes that form a loop when the current node is present in the AS path. Otherwise, the router's policy, per its configuration, is applied.

Figure 5 shows an example, where a.lp and a.path denote components of an attribute a = (lp, tags, path). Assume that in this network b_2 prefers going through a to reach destination d and that this policy is achieved by configuring a to add tag 1 to outgoing messages and configuring b_2 to prefer this tag. The configuration-driven part of the transfer function is shown in the boxes for routers a and b_2. Router a adds the tag 1 to attributes it exports; and b_2 checks for this tag, and if present, assigns a higher (better) local preference value than the default value (100), which ensures that b_2 prefers to go through a. The arrows in the figure indicate the final

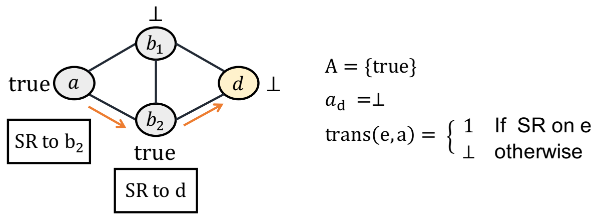


Figure 6: Modeling Static routing with SRP.

forwarding behavior of this network, and a solution labeling $\mathcal{L}$ is shown next to each node.

Static routing. Operators configure static routes that describe which interface to use for a given destination. Figure 6 shows an example where routers a and b_2 are configured with static routes. We model static routing using the set of attributes $A = \{true\}$ which indicates the presence of a static route. Since there is only one attribute, the comparison relation is trivially empty. The transfer function does not depend on the neighbor at all; it returns *true* if there is a static route configured locally along an edge and $\perp$ otherwise.

4 EFFECTIVE ABSTRACTIONS

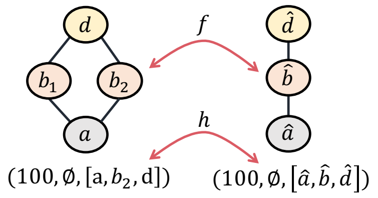
We now build on SRPs to describe the theory and implications of effective network abstractions.

Network abstractions. We start by formalizing network abstractions. A network abstraction relates two SRPs—a concrete $SRP = (G, a_d, A, \prec, \text{trans})$ and an abstract $\widehat{SRP} = (\widehat{G}, \widehat{a_d}, \widehat{A}, \widehat{\prec}, \widehat{\text{trans}})$—using a pair of functions (f, h). The topology function $f : V \to \widehat{V}$ maps each concrete graph node to an abstract graph node, and the attribute function $h : A \to \widehat{A}$ maps each concrete attribute to an abstract one. For convenience, we will write $u \mapsto \widehat{u}$ to mean $f(u) = \widehat{u}$, and $a \mapsto \widehat{a}$ to mean $h(a) = \widehat{a}$. We also freely apply f to edges and paths: given an edge $e = (u, v)$, $f(e)$ means $(f(u), f(v))$; given a path $u_1, \ldots, u_n$, $f(u_1, \ldots, u_n)$ means $f(u_1), \ldots, f(u_n)$.

Attribute abstraction allows the set of attributes to differ between the concrete and abstract networks. This ability may be used to convert attributes with concrete nodes into those with related abstract nodes. For example, in the BGP network in Figure 7, f maps b_i nodes to the abstract node $\widehat{b}$, while h maps the concrete AS path to its abstract counterpart.

4.1 Effective Abstraction Conditions

In a network abstraction, f and h can be arbitrary functions, but we are interested only in abstractions that preserve the control plane behavior of the concrete network. An *effective* abstraction satisfies a set of relatively easy-to-check conditions that imply CP-equivalence. These conditions, listed in the middle right of Figure 4, are restrictions on the topology function f and the attribute function h.



d
b_1
b_2
a
f
h
$\hat{d}$
$\hat{b}$
$\hat{a}$
$(100, \emptyset, [a, b_2, d])$
$(100, \emptyset, [\hat{a}, \hat{b}, \hat{d}])$

Figure 7: Example abstraction for a BGP network.

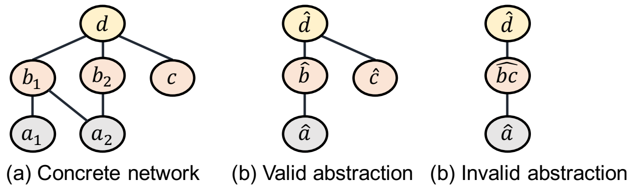


Figure 8: Topology abstraction.

Topology abstraction conditions. Effective topology functions obey two conditions. First, they preserve the identity of the destination node (*dest-equivalence*). That is, the concrete destination node, and only this node, should be mapped to the abstract destination: $d \mapsto \widehat{d}$, $d' \not\mapsto \widehat{d}$. Second, the topological mapping as a whole must be a (forall-exists) ∀∃−abstraction. A ∀∃−abstraction (both ∀∃−abstraction1 and ∀∃−abstraction2) demands that: (1) for every concrete edge (u, v) there is a corresponding abstract edge $(\widehat{u}, \widehat{v})$ and (2) for every abstract edge $(\widehat{u}, \widehat{v})$, *all* concrete nodes u where $u \mapsto \widehat{u}$ must have an edge to *some* concrete node v where $v \mapsto \widehat{v}$. Figure 8 shows an example of both a valid and invalid ∀∃−abstraction. The abstraction on the right is invalid because c does not have an edge to either a_1 or a_2 despite there being an edge between $\widehat{bc}$ and $\widehat{a}$ in the abstract network.

Attribute abstraction conditions. The first conditions for attribute abstraction, *drop-equivalence* and *orig-equivalence*, state that the abstraction function must preserve the "no route" and the destination attributes: $h(\perp) = \perp$ and $h(a_d) = \widehat{a_d}$. An abstraction must also preserve the comparison relation's attribute ordering (*rank-equivalence*). Finally, an abstraction must preserve the transfer function (*transfer-equivalence*), that is, applying the concrete transfer function and abstracting the resulting attribute should be the same as abstracting the attribute first, and then applying the abstract transfer function. A critical aspect here is that, unlike CP-equivalence, which is a network-wide property, transfer-equivalence is a simple, local property that can be evaluated efficiently by comparing the transfer functions.

4.2 Effectiveness implies CP Equivalence

We are now ready to prove that effective abstractions guarantee CP-equivalence in two steps. First, we demonstrate that effective abstractions are *label-equivalent* (Figure 4). In other words, for each solution $\mathcal{L}$ to SRP, there exists a corresponding solution $\widehat{\mathcal{L}}$ to the abstract $\widehat{SRP}$ (*i.e.*, whenever $\mathcal{L}$ labels u with a, $\widehat{\mathcal{L}}$ labels $f(u)$ with $h(a)$), and vice-versa. Next, we show that given related labellings, the final control plane behaviors are also related, *i.e.*, they are equivalent with respect to forwarding (*fwd-equivalent* as defined in Figure 4).

Our proof depends on the structure of the SRPs and their solutions. In particular, when the SRP nodes dynamically transmit information to one another, we would like to be able to carry out the proof using induction. However, we cannot do that if the SRP solutions contain loops, as the induction would not be well-founded. Fortunately, most broadly-used dynamic routing protocols are loop-free by design. We will consider the simpler case of static routes, which can be configured to create loops, separately.

Theorem 4.1. *Any solution $\mathcal{L}$ to a well-formed, loop-free SRP will form a DAG rooted at the destination d.*

Using this property of stable solutions, we can prove that for any concrete solution $\mathcal{L}$, there is an abstract solution $\widehat{\mathcal{L}}$ such that the solutions are label- and fwd-equivalent (and vice-versa). The proof goes in two steps. First, we prune the graph to include only edges in $\mathcal{L}$ or $\widehat{\mathcal{L}}$ that are involved in forwarding. Within such subgraphs, we can show by induction on the length of the forwarding paths that the subgraphs satisfy label-equivalence and fwd-equivalence. It is then easy to come to our desired conclusion by showing that adding the removed edges back in does not affect the stable solution of either the concrete or the abstract graph.

Theorem 4.2. *A well-formed, loop-free SRP and its effective abstraction $\widehat{SRP}$ are label- and fwd-equivalent.*

Using Theorem 4.2, we may also conclude that any effective abstractions of common protocols, which produce loop-free routing, are CP-equivalent. However, effectiveness requires transfer-equivalence, which as mentioned previously commonly does not hold for BGP. That makes it impossible to obtain effective abstractions for BGP networks. In the next subsection, we address this shortcoming by defining another kind of abstraction that is applicable for BGP.

Static routing. Networks with static routes are not necessarily loop-free. (The presence of a loop would clearly be a bug, but we must be sure our theory is sound in such a situation so we can use it to detect inadvertent bugs caused by misconfiguration of static routes.) Fortunately, due to the simple nature of static routing—static routes do not depend on other routes learned from neighbors—we can prove its correctness independently.

Theorem 4.3. *A self-loop-free SRP and $\widehat{SRP}$ for static routing with an effective abstraction will have fwd-equivalence.*

4.3 BGP with Loop Prevention

We model BGP using an abstraction: $h((\text{lp}, \text{tags}, \text{path})) = (\text{lp}, \text{tags}, f(\text{path}))$. BGP's loop-prevention is problematic here because it depends on the actual concrete path used, which implies that two concrete nodes x and y with syntactically identical configurations will actually have different transfer functions and violate transfer-equivalence. Node x will reject paths that have gone through x but not y, and node y will reject paths that have gone through y but not x. If we were somehow able to abstract away loop prevention, we could attempt to have topology abstractions for BGP that are transfer-equivalent. This observation motivates the additional properties laid out for BGP in Figure 4.

BGP-effective abstractions. For BGP, we require dest-, drop-, orig- and rank-equivalence as for ordinary effective abstractions. However, as opposed to a $\forall\exists-$abstraction, we require a slightly stronger (forall-forall) $\forall\forall-$abstraction. This constraint requires that there is an abstract edge between $\widehat{u}$ and $\widehat{v}$ if and only if there is a concrete edge between u and v. This strong condition on the network topology allows us to get away with a weaker condition than transfer-equivalence: we relax the transfer-equivalence condition to what we call transfer-approx. The latter condition is similar to transfer equivalence, except it ignores differences caused by BGP loop-prevention. Formally, it is specified as:

$$\forall e, a.\ e = (u, v) \land v \notin a.\text{path} \Rightarrow$$
$$h(\text{trans}(e, a)) = \widehat{\text{trans}}(f(e), h(a))$$

Bounded behaviors. Now, given a BGP-effective abstraction, we know that, when loop-prevention happens, there may be differences between the forwarding behaviors of different concrete nodes even when they have identical configurations. Fortunately, we can bound the number of different behaviors that can arise dynamically, and, moreover, we can infer that bound directly from the configurations.

First, let $\mathcal{B}_{\mathcal{L}}(\widehat{u})$ be the set of possible behaviors of concrete nodes related to abstract node $\widehat{u}$. Second, let prefs(v) be the set of BGP local-preference values that may be assigned to an announcement at node v. For example, if a configuration explicitly sets the local-preference value to 200 or 300 depending on the route, and 100 is the default local preference, then the set prefs$(v) = \{100, 200, 300\}$. With these definitions in hand, we can prove the following theorem.

Theorem 4.4. *If a well-formed SRP and $\widehat{SRP}$ for BGP has an $\forall\forall-$abstraction and is transfer-approx, then for all solutions $\mathcal{L}$ to SRP, and all abstract nodes $\widehat{u} \in \widehat{V}$, $|\mathcal{B}_{\mathcal{L}}(\widehat{u})| \leq |\text{prefs}(\widehat{u})|$.*

Abstraction refinement. A bound on the the number of behaviors for nodes in BGP lets us refine an abstraction by splitting apart abstract nodes into enough cases to recover CP-equivalence. We now formalize this intuition.

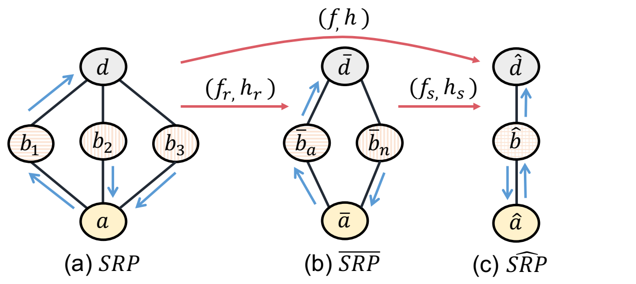


Figure 9: Abstraction refinement for Figure 2(a).

Suppose we are given an $SRP = (G, A, a_d, \prec, \text{trans})$ for BGP and its abstract version $\widehat{SRP} = (\widehat{G}, \widehat{A}, \widehat{a_d}, \widehat{\prec}, \widehat{\text{trans}})$, which are self-loop-free and created from a $\forall\forall-$abstraction (f, h). We define a new abstraction $\overline{SRP} = (\overline{G}, \overline{A}, \overline{a_d}, \overline{\prec}, \overline{\text{trans}})$ obtained by splitting up each node $\widehat{v}$ into $|\text{prefs}(\widehat{v})|$ copies of the node. We can view the mapping from SRP to $\widehat{SRP}$ as the composition of two abstractions (f_r, h_r) from SRP to $\overline{SRP}$, and (f_s, h_s) from $\overline{SRP}$ to $\widehat{SRP}$, where the comparison and transfer functions for $\overline{SRP}$ are copied from $\widehat{SRP}$. Given a new abstraction (f_r, h_r) where $f_r : V \rightarrow \overline{V}$ and $h_r : A \rightarrow \overline{A}$, we say (f_r, h_r) *refines* (f, h), written as $(f_r, h_r) \sqsubseteq_{(f_s, h_s)} (f, h)$ if f_r is an *onto* function, and $f = f_s \circ f_r$ and $h = h_s \circ h_r$.

We now show that there is a bisimulation between the solutions $\mathcal{L}$ and $\overline{\mathcal{L}}$ as before. However, whereas the abstraction mapping f was known in advance, the refined mapping f_r may change depending on the particular solution $\mathcal{L}$. For example, Figure 9(a) shows one of three possible forwarding behaviors for the network. As discussed earlier, with a different message arrival timing, other solutions would have emerged. Depending upon this solution, different nodes, *e.g.* $\{b_1, b_2\}$ or $\{b_1, b_3\}$ would be mapped to $\overline{b_n}$. We do not know which concrete nodes are mapped to which abstract nodes, but we do know that the abstraction has sufficiently many nodes to characterize all possible behaviors.

Theorem 4.5. *Suppose we have well-formed SRP, $\widehat{SRP}$, and $\overline{SRP}$ for BGP with an effective abstraction (f, h). There is a solution $\mathcal{L}$ to SRP iff there is a solution $\overline{\mathcal{L}}$ to $\overline{SRP}$, such that there exists a refinement $(f_r, h_r) \sqsubseteq_{(f_s, h_s)} (f, h)$ and $\mathcal{L}$ and $\overline{\mathcal{L}}$ are label- and fwd-equivalent.*

A key difference between Theorem 4.2 and Theorem 4.5, is that the forwarding paths between the concrete network (SRP) and the refined network ($\overline{SRP}$) will only be equivalent with respect to the original abstract network ($\widehat{SRP}$). For example, in Figure 9(a), if we want to check that b_2 and b_3 forward along a path that satisfies some property p, then we can not check it against only $\overline{b_a}$ in Figure 9(b). Rather, we have to check it against $\overline{b_n}$ as well because there is another stable solution where the roles of b_a and b_n are reversed.

4.4 Properties preserved

As a consequence of CP-equivalence, for a solution $\mathcal{L}$ to the concrete network, there exists a path $s = x_1, \ldots, x_k$ where the network forwards along s with labels $\mathcal{L}(x_1), \ldots, \mathcal{L}(x_k)$ iff for some solution $\widehat{\mathcal{L}}$ to the abstract network, there is a path $f(s)$ where the abstract network forwards along $f(s)$ with labels $\widehat{\mathcal{L}}(f(x_1)), \ldots, \widehat{\mathcal{L}}(f(x_k))$. Concretely, one can check that any of the following properties hold on small abstract networks and be sure the concrete counterpart satisfies the property as well.

- **Reachability:** $f(u)$ can reach $f(v)$ in the abstract network iff u can reach v in the concrete network.
- **Path Length:** All paths between $f(u)$ and $f(v)$ have length n iff all paths between u and v have length n.
- **Black Holes:** Path s in the concrete network ends with label $\perp$ iff path $f(s)$ ends with $\perp$ in the abstract network.
- **Multipath Consistency:** Traffic sent from $f(u)$ is reachable along some path to $f(v)$ but dropped along another path iff traffic from u is reachable along some path to v and dropped along another path.
- **Waypointing:** Traffic will be waypointed through one of $\{f(w_1), \ldots, f(w_n)\}$ in the abstract network iff it will go through one of $\{w_1, \ldots, w_n\}$ in the concrete network.
- **Routing Loops:** There is a routing loop in the abstract network iff there is one in the concrete network.

Convergence. The concrete network necessarily diverges (has no stable solution) iff the abstract network necessarily diverges. To see why, suppose the concrete network had no stable solution, but the abstract network had a stable solution. This would violate CP-equivalence, since each abstract solution has a corresponding concrete solution. Similarly, the concrete network can converge (has some stable solution) iff the abstract network can converge. However, CP-equivalence alone does not guarantee that networks that might converge or might diverge, like the naughty gadget in BGP [22], will necessarily reduce to an abstract network that may diverge.

On the other hand, effective abstractions are stronger than (imply) CP-equivalence. We postulate (but have not proven) that an effective abstraction is sufficient to preserve convergence. For example, it would appear that the concrete network will have a dispute wheel [22] (the lack of which is sufficient condition for convergence safety and robustness) iff the abstract network has a dispute wheel (the nodes in concrete network forming a dispute wheel will induce a dispute wheel in their abstract counterpart).

4.5 Properties not preserved

While effective abstractions preserve the nature of forwarding paths, they do not, in general, preserve the number of paths or the number of neighbors. Indeed, that is the point—effective abstractions usually reduce the number of paths and neighbors to speed analysis. Consequently, we cannot reason faithfully about properties such as fault tolerance, load balancing, or any QoS properties. For instance, in the abstract network, a single link failure may partition a network whereas in the concrete network, there may exist two or more link-disjoint paths between all pairs of nodes, allowing the concrete network to tolerate any single failure.

5 ABSTRACTION ALGORITHM

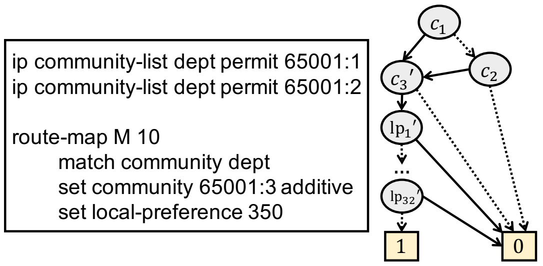
Earlier sections described the conditions under which an abstraction will preserve CP-equivalence, but they give no insight into how one might compute such an abstraction. In this section, we describe an algorithm that computes an abstraction directly from a set of router configurations.

5.1 Algorithm Overview

Our algorithm starts with the following observations. The key requirement for computing an effective abstraction is to ensure that we satisfy each required condition in Figure 4. Some conditions such as orig-equivalence ($h(a_d) = \widehat{a_d}$), drop-equivalence ($h(a) = \perp \iff a = \perp$) and rank-equivalence ($a \prec b \iff h(a) \mathrel{\widehat{\prec}} h(b)$) depend only on the particular protocol and choice of h. By fixing h in advance for each protocol similar to those used in Figures 5 and 6, we can guarantee that these conditions hold regardless of the configurations. Other conditions such as dest-equivalence and $\forall\exists$−abstraction depend on the topology, but not the policy embedded in configurations.

Transfer-equivalence: $h(\mathsf{trans}(e, a)) = \widehat{\mathsf{trans}}(f(e), h(a))$ is the only condition that depends on user-defined policy. Suppose two concrete edges e_1 and e_2 are mapped together by the topology function f. We would have $h(\mathsf{trans}(e_1, a)) = \widehat{\mathsf{trans}}(f(e_1), h(a)) = \widehat{\mathsf{trans}}(f(e_2), h(a)) = h(\mathsf{trans}(e_2, a))$. One simple way to ensure that this equality holds is to only combine together nodes with the same transfer function. In our example, $\mathsf{trans}(e_1, a) = \mathsf{trans}(e_2, a)$ would suffice to allow e_1 and e_2 to map to the same abstract edge.

Based on the observations above, we fix h; our remaining task is to find a suitable f that satisfies the topology abstraction requirements and only maps together edges with equivalent transfer functions (for the destination d). We find such a function f using an algorithm based on abstraction refinement. We start with the coarsest possible abstraction where there is a single abstract destination node $\widehat{d}$ and one other abstract node for all other concrete nodes, and while the abstraction violates $\forall\exists$−abstraction or transfer-equivalence, we refine it by breaking up the problematic abstract node into multiple abstract nodes.



```
ip community-list dept permit 65001:1
ip community-list dept permit 65001:2

route-map M 10
      match community dept
      set community 65001:3 additive
      set local-preference 350
```

Figure 10: BDD for a BGP policy on an interface.

For efficiency, before abstraction refinement, we process router configurations in two different ways.

1. Destination Equivalence Classes (ECs). In our theoretical account of routing, each SRP contains a single destination. However, in practice, configurations contain routing information for many destinations simultaneously. Because announcements for (most) destinations do not interact with one another, we can partition the network into equivalence classes based on where destinations are rooted. Each class has a collection of destination IP ranges and destination node(s). This partitioning allows us to build one abstraction per class instead of one per address. To partition the network into equivalence classes, we use a prefix-trie data structure where the leaves of the trie contain a set of destination nodes.

2. Encoding transfer function using BDDs. In order to efficiently find all interfaces that have equivalent transfer-functions for a given destination (class), we use Binary Decision Diagrams (BDDs) [9] to represent the routing policy for each interface. BDDs can compactly represent Boolean functions and are a canonical representation for such functions. Memoization combined with uniqueness of the representation means that two BDDs are semantically-equivalent iff their pointers are the same. This turns checking equivalence of any two transfer functions into an $O(1)$ operation after their BDDs are constructed.

As an example, consider the BGP routing policy in Figure 10. The policy checks if either the 65001:1 or 65001:2 community is attached to an inbound route advertisement. If so, it adds the 65001:3 community and updates the local preference to 350. Each node in the BDD represents a boolean variable used to represent state in the advertisement. Primed variables represent output values after applying updates to the advertisement. A solid arrow means the value is true, while a dashed arrow means the value is false. There are two leaf values: 0 and 1 which represent false and true, respectively. Any path from the BDD root to 1 represents a valid input-output relation. If c_1, the variable representing community 65001:1 is true, then the resulting advertisement will have c_3' true (65001:3 attached), and will have a local preference for the 32 bit value representation of 350.

Algorithm 1 Compute abstraction function f

```
1:  procedure FindAbstraction(Graph G, Bdds bdds)
2:      Specialize(bdds, G.d)
3:      f ← UnionSplitFind(G.V)
4:      Split(f, {$G.d$})
5:      while True do
6:          $\widehat{V}$ ← Partitions(f)
7:          for $\widehat{u}$ in $\widehat{V}$ do
8:              if $|\widehat{u}| \leq 1$ then continue
9:              Refine(G, bdds, f, $\widehat{u}$, |prefs($\widehat{u}$)|)
10:         $\widehat{V}'$ ← Partitions(f)
11:         if $|\widehat{V}| = |\widehat{V}'|$ then break
12:     return SplitIntoBGPCases(f)
13:
14: procedure Refine(G, bdds, f, $\widehat{u}$, numPrefs)
15:     map ← CreateMap
16:     for $u \in \widehat{u}$ do
17:         for $e = (u, v) \in$ G.E do
18:             pol ← Get(bdds, e)
19:             n ← (numPrefs > 1 ? $v : f(v)$)
20:             map[u] ← map[u] ∪ { (pol, n) }
21:     for us ∈ GroupKeysByValue(map) do
22:         Split(f,us)
```

5.2 The Algorithm

Algorithm 1 lists the steps used to compute the abstraction function f given graph (G) and a collection of BDDs ($bdds$). The first step is to specialize the $bdds$ to the particular destination $G.d$ (line 2). We use a union-split-find data structure to maintain a collection of disjoint sets of concrete nodes that represent the abstract nodes in the network. One of the first steps is to split the collection of sets so that $G.d$ becomes its own abstract node (line 4) and every other concrete node remains as a single other abstract node. Next, it repeatedly tries to refine the abstraction while it is not a effective abstraction. The algorithm iterates over each current abstract node (set of concrete nodes). If the abstract node is already fully concrete (line 8), then it continues, otherwise it refines the abstraction. Refine iterates over each concrete node u in the abstract node $\widehat{u}$ and each edge from u to v, and builds a map from u to a set of pairs of the BDD policy along edge (u, v) and the neighboring node (line 20)—either the concrete neighbor (for ∀∀−abstraction) or the abstract neighbor (for ∀∃−abstraction). Finally, we group entries of the map (us) by those values that have the same pairs of policies and neighbors, and then refine the abstraction by these groups (line 22). This step ensures that groups of devices that have different transfer functions or policies to different neighbors are separated in the next iteration of the algorithm.

Figure 11 shows the output of the algorithm on a BGP-based fattree network with two different routing policies. In one case, the network uses shortest (AS) path routing, and in the second case, the middle-tier of routers prefer to route via

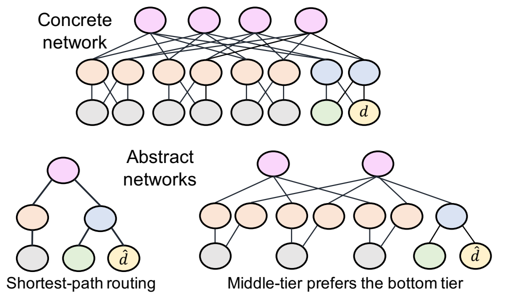


Figure 11: Abstractions for a network running BGP on a fattree topology using different policies.

the bottom tier. The abstract network is bigger in the second case to capture the greater number of possible forwarding behaviors of the middle-tier routers.

6 PRACTICAL EXTENSIONS

Multiple Protocols. Although the stable routing problem is framed in terms of the behavior of a particular protocol, devices in practice often run multiple protocols at once. One can build a new SRP to model these interactions. For example, if a network runs both OSPF and eBGP, then the SRP could use attributes of the form $A = A_{BGP} \times A_{OSPF} \times A_{RIB}$. That is, track both OSPF and BGP, as well as A_{RIB}, which represents the main RIB that carries the best route (based on administrative distance) between the various protocols and records what protocol was chosen. Following ideas from Batfish [19], we model route redistribution, where routes from one protocol are injected into another, via the transfer function. For instance, if OSPF routes are redistributed into BGP, then BGP will allow routes from A_{RIB} even when they are from OSPF.

Access Control Lists. While ACLs do not affect control plane routing information, they can prevent traffic from being forwarded out an interface. For this reason, we conservatively consider the ACL to be part of the transfer function, which gets captured in the BDD, so that nodes will only be abstracted together if they have the same ACLs with respect to destination d. This ensures that the fwd-equivalence property will remain valid.

iBGP. iBGP is a complicated protocol that recursively routes packets for eBGP by communicating them over an IGP path. If there is a valid abstraction for both the IGP and for eBGP, and there is no ACL in the network that blocks iBGP loopback addresses, then multiple iBGP neighbors can be compressed together. This is because (1) both iBGP neighbors will be sent the same eBGP routes from neighbors, (2) these advertisements will have the same IGP cost metric (since they must be symmetric with respect to the IGP as well), and (3) although the iBGP neighbors may have an edge between them, potentially violating the self-loop-free requirement, this edge is never used since iBGP does not re-advertise routes learned over iBGP to other iBGP neighbors.

7 IMPLEMENTATION

We implemented our network abstraction algorithm in a tool called Bonsai. It uses the Batfish [19] network analysis framework to convert network configurations into a vendor-independent intermediate representation. Bonsai operates over this vendor-independent format to create a network abstraction in the form of a smaller, simpler collection of vendor-independent configurations. Tools built using this framework, such as Batfish and Minesweeper, can then work with the smaller configurations to speed up their analysis.

We use the Javabdd [43] library to encode router-level import and export filters, as well as access control lists (ACLs) as BDDs. Because Bonsai creates abstract networks per destination EC, and such ECs are disjoint,our implementation is able to generate abstract networks and check their properties in parallel. We only generate abstract networks for destination ECs that are relevant for a query. For example, checking port-to-port reachability would typically only require generating a single abstract network for one EC.

8 EVALUATION

We evaluate Bonsai using a collection of synthetic and real networks. We aim to answer the following questions: (i) can Bonsai scale to large networks? (ii) can its algorithm effectively compress networks? and (iii) can the abstract, compressed networks be speed network analysis?

Networks studied. We study three types of synthetic network topologies: Fattree [2], Ring, and Full-mesh. Each such network uses eBGP to perform shortest path routing along with destination-based prefix filters to each destination. These networks are highly symmetric by design and we use them to characterize compression as a function of network topology and size. For each topology type, we scale the size and measure the effectiveness and cost of compression.

While the synthetic networks focus on the effect of topology on compression, in practice, most networks do not have perfect symmetry. For this reason, we study operational networks of two different corporations. The first is a datacenter network with 197 routers organized into multiple clusters, each with a Clos-like topology (rather than a single, large Clos-like topology). The network primarily uses eBGP and static routing, with each router running as its own AS using BGP private AS numbers. It also makes extensive use of route filters, ACLs, and BGP communities. All together, it has over 540,000 lines of configuration. Although there are less than

Topology	Nodes / Edges	Abs. Nodes / Edges	Compression ratio	Num ECs	BDD time	Compression time (per EC)
(a) Synthetic networks						
Fattree	180 / 2124	6 / 5	30× / 424.8×	72	0.36	0.09
	500 / 9100	6 / 5	83.33× / 1820×	200	1.29	0.26
	1125 / 29475	6 / 5	187.5× / 5895×	450	7.87	0.75
Ring	100 / 100	51 / 50	1.96× / 2×	100	0.14	0.08
	500 / 500	251 / 250	1.99× / 2×	500	0.33	2.29
	1000 / 1000	501 / 500	2× / 2×	1000	0.34	12.26
Full Mesh	50 / 1225	2 / 1	25× / 1225×	50	0.18	0.07
	150 / 4950	2 / 1	75× / 4950×	150	1.11	0.34
	250 / 31125	2 / 1	125× / 31125×	250	3.31	5.48
(b) Real networks						
Data center	197 / 16091	30.2 ± 2.2 / 143.6 ± 18.6	6.6× / 112×	1269	132.28	15.51
WAN	1086 / 5430	209.4 ± 36.5 / 759.4 ± 129.2	5.2× / 7.2×	845	11.35	1.83

Table 1: Compression results for synthetic and real networks. All times are in seconds.

200 routers in the network, there are over 16,000 physical and virtual interfaces in the network.

The second operational network is a wide-area network (WAN) with 1086 devices, which are a mix of routers and switches. The network uses a eBGP, iBGP, OSPF, and static routing, and consists of over 600,000 lines of configuration.

Synthetic network results. Table 1(a) shows the results of running Bonsai on the synthetic networks. All experiments were run on an 8-core Macbook Pro with an Intel i7 processor and 8GB of RAM. For each synthetic network, Bonsai is able to compress the network quickly. For instance, the largest Fattree topology with 1125 nodes takes around 7.9 seconds to build the BDD data structures and an average of of .75 seconds per EC to compute the abstract network for the 450 ECs. Because equivalence classes are processed in parallel, it takes under a minute to abstract this network. The compressed network size computed is 6 nodes.

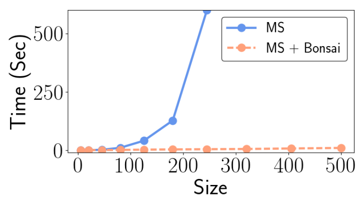
For the Fattree and Full-mesh topologies, the compressed network size stays constant as the concrete network grows. For the ring topology, the compressed network size does grow with the size of the network, and in particular, grows with the diameter of the network. This is necessary since the abstraction must preserve path length. Computing an abstraction for the ring topologies is more expensive because the compression algorithm is only able to split out a single new abstract role with each iteration.

Bonsai's compression has a large effect on network analysis time. Figure 12 shows the total verification time to check an all-pairs reachability query compared to topology size for each type of synthetic network using Minesweeper [6]. We use a timeout of 10 minutes. The verification time for abstract networks includes the time used to partition the network, build the BDDs, and compute the compressed network. In all cases, abstraction significantly speeds up verification even when taking into account the time to run Bonsai. Abstracting the Full-mesh topology ran out of memory beyond a certain point, due to the density of the topology.

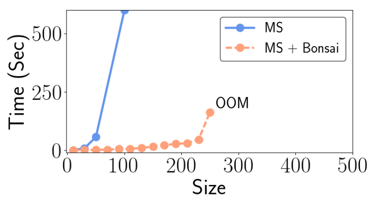
Real network results. For both networks, we first computed the BDDs and see how many devices have identical transfer functions from their configurations. In the datacenter network, we initially found that there were 112 unique "roles" (set of policies) among the 197 routers. However, many of these differences could be attributed to BGP community values that were attached to routers, but then never matched on in any configuration file. To account for these differences, we use the abstraction function for BGP: $h(\text{lp}, \text{tags}, \text{path}) = (\text{lp}, \text{tags} - \{\text{unused}\}, f(\text{path}))$, which ignores differences from such irrelevant tags. With this abstraction function, we find that there are only 26 unique "roles" among the 197 routers. Further, most of the differences are due to differences in static routes in the configurations. Without static routes, there would only be 8 unique roles. Table 1 (b) shows the compression results from this network. It takes just over 2 minutes to compute the BDDs and roughly 15 seconds on average to compute a good abstraction per EC. This time is mainly due to the huge number of virtual interfaces. The average compressed network size is around 30 nodes (a 6.6x reduction), and around 132 edges (a 112x reduction).

For the WAN, we found 137 unique "roles" among the 1086 devices. Many of the differences are from neighbor-specific, prefix-based filters and ACLs. It takes around 11 seconds to compute the BDDs for the network, and under 2 seconds per EC to compute a good abstraction. The average compressed size achieves a 5.2x reduction in the number of nodes and a 7.4x reduction in the number of edges.

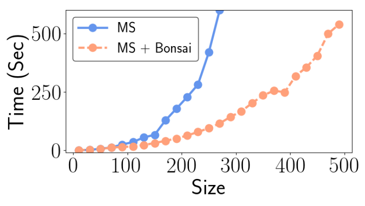
Finally, to test the effectiveness of Bonsai at facilitating scalable analysis of real networks, we run a reachability query between two devices in Batfish, both with and without abstraction. Batfish first simulates the control plane to produce the data plane and then uses NoD [32] to compute



(a) Fattree



(b) Full Mesh



(c) Ring

Figure 12: Minesweeper (MS) verification time with and without abstraction for an all-pairs reachability query.

all possible packets that can traverse between source and destination nodes. With Bonsai, it takes 77 seconds to complete the query. Without it, the query did not complete and gave an out-of-memory error after running for over an hour.

9 RELATED WORK

Network verification. The field of network verification may be split into data plane verification [8, 25–27, 29, 32, 34, 44] and control plane verification [6, 15, 16, 19, 20, 42], with our work sitting in the latter camp. However, Bonsai is orthogonal to, and synergistic with, most of this reseach as it compresses networks and leaves the analysis to other tools, which typically operate much more quickly over the compressed network. Bonsai works because large networks typically contain symmetries, an observation made and exploited by Plotkin *et al.* [36], though Plotkin *et al.* focus on data plane properties whereas we focus on control plane properties. The only other control plane compression work we are aware focuses on compressing BGP networks using local rewrites to preserve convergence properties [41]. In contrast, we introduce the SRP model to compress networks running a wide-variety of protocols using both local and some non-local (BGP splitting) rewrites. We aim to preserve forwarding properties so network administrators can test for reachability, access control and other path-based properties rather than convergence.

Control plane models. A formal model of network control plane planes lies at the heart of our work. Many prior works have developed such models to describe formally the computation of routing protocols, their safety criteria, or to generalize their computation [22, 23, 38]. Our model, SRP, is inspired by Griffin *et al.*'s stable paths problem (SPP) which described control plane solutions computed by path vector protocols [22]. While both models describe stable solutions, SRP formalizes device-level processing of routing information instead of end-to-end paths. This difference allows it to capture a broader range of control plane features.

SRPs are similar to routing algebras [23, 38], though we have simplified our presentation slightly by allowing graph edges to stand in for the labels used in routing algebras. However, the more significant difference is that while routing algebras have been used to study convergence properties, which are independent of network topology, we study topology-dependent properties such as reachability, and developed compression algorithms that preserve such properties.

Abstractions in verification. Conservative abstractions are the mainstay of program verification in various forms such as loop invariants [18, 24], abstract interpretation [13], and counterexample guided abstraction refinement [4, 11, 12]. These abstractions enable sound analysis for verification problems that are often undecidable or intractable. Tighter abstractions based on symmetry and bisimulations have also been used successfully to scale model checking [10, 14]. We build on these foundations to seek useful abstractions for compressing networks that preserve CP-equivalence.

10 CONCLUSION

Recently, researchers have made great progress in control plane analysis, using a variety of techniques ranging from simulation to verification. But the scale and complexity of real networks often renders such techniques computationally expensive or even intractable. To accelerate analysis, our Bonsai tool automatically compresses a network and its configurations by eliminating any structural symmetries. Bonsai is based on a theory of control plane equivalence of two networks and an efficient compression algorithm. We show that it scales well and effectively compresses real networks.

Acknowledgements. We would like to thank the ACM SIGCOMM reviewers and our shepherd Laurent Vanbever, whose feedback helped improve this paper. This work was supported in part by NSF Grants 1703493 and 1525936, and gifts from Cisco and Facebook. Any opinions, findings, and conclusions expressed herein are those of the authors and do not necessarily reflect those of the NSF, Cisco or Facebook.

REFERENCES

[1] Control plane compression: Extended version. https://scholar.princeton.edu/sites/default/files/rbeckett/files/paper_0.pdf.

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

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

[4] T. Ball, R. Majumdar, T. D. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. In *PLDI*, pages 203–213, 2001.

[5] L. Bauer, S. Garriss, and M. K. Reiter. Detecting and resolving policy misconfigurations in access-control systems. *ACM Trans. Information and System Security*, 14(1), June 2011.

[6] R. Beckett, A. Gupta, R. Mahajan, and D. Walker. A general approach to network configuration verification. In *SIGCOMM*, August 2017.

[7] BGPMon. http://www.bgpmon.net/.

[8] N. Bjørner, G. Juniwal, R. Mahajan, S. A. Seshia, and G. Varghese. *ddNF: An Efficient Data Structure for Header Spaces*. November 2016.

[9] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[10] E. M. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In *Computer Aided Verification, CAV*, 1993.

[11] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, pages 154–169, 2000.

[12] E. M. Clarke, D. Kroening, N. Sharygina, and K. Yorav. Predicate abstraction of ANSI-C programs using SAT. *Formal Methods in System Design*, 25(2-3):105–127, 2004.

[13] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.

[14] E. A. Emerson and A. P. Sistla. Symmetry and model checking. In *Computer Aided Verification, 5th International Conference, CAV*, 1993.

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

[16] N. Feamster and H. Balakrishnan. Detecting BGP configuration faults with static analysis. In *NSDI*, May 2005.

[17] N. Feamster and J. Rexford. Network-wide prediction of BGP routes. *IEEE/ACM Trans. Networking*, 15(2), 2007.

[18] R. W. Floyd. Assigning meanings to programs. In *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32, 1967.

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

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

[21] J. Godfrey. The summer of network misconfigurations. https://blog.algosec.com/2016/08/business-outages-caused-misconfigurations-headline-news-summer.html, 2016.

[22] T. G. Griffin, F. B. Shepherd, and G. Wilfong. The stable paths problem and interdomain routing. *IEEE/ACM Trans. Networking*, 10(2), 2002.

[23] T. G. Griffin and J. L. Sobrinho. Metarouting. In *SIGCOMM*, pages 1–12, August 2005.

[24] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580, 1969.

[25] A. Horn, A. Kheradmand, and M. Prasad. Delta-net: Real-time network verification using atoms. In *NSDI*, March 2017.

[26] P. Kazemian, M. Chang, H. Zeng, G. Varghese, N. McKeown, and S. Whyte. Real time network policy checking using header space analysis. In *NSDI*, pages 99–112, April 2013.

[27] P. Kazemian, G. Varghese, and N. McKeown. Header space analysis: Static checking for networks. In *NSDI*, April 2012.

[28] Z. Kerravala. What is behind network downtime? proactive steps to reduce human error and improve availability of networks. https://www.cs.princeton.edu/courses/archive/fall10/cos561/papers/Yankee04.pdf, 2004.

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

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

[31] H. H. Liu, Y. Zhu, J. Padhye, J. Cao, S. Tallapragada, N. P. Lopes, A. Rybalchenko, G. Lu, and L. Yuan. Crystalnet: Faithfully emulating large production networks. In *SOSP*, pages 599–613, March 2017.

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

[33] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *SIGCOMM*, August 2002.

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

[35] J. Networks. As the value of enterprise networks escalates, so does the need for configuration management. https://www-935.ibm.com/services/au/gts/pdf/200249.pdf, 2008.

[36] G. D. Plotkin, N. Bjørner, N. P. Lopes, A. Rybalchenko, and G. Varghese. Scaling network verification using symmetry and surgery. In *POPL*, January 2016.

[37] S. Prabhu, A. Kheradmand, B. Godfrey, and M. Caesar. Predicting network futures with plankton. In *Proceedings of the First Asia-Pacific Workshop on Networking*, APNet'17, pages 92–98, August 2017.

[38] J. a. L. Sobrinho. An algebraic theory of dynamic network routing. *IEEE/ACM Trans. Netw.*, 13(5):1160–1173, October 2005.

[39] R. Stoenescu, M. Popovici, L. Negreanu, and C. Raiciu. Symnet: Scalable symbolic execution for modern networks. In *SIGCOMM*, 2016.

[40] Y. Sverdlik. 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, 2012.

[41] A. Wang, A. J. T. Gurney, X. Han, J. Cao, B. T. Loo, C. Talcott, and A. Scedrov. A reduction-based approach towards scaling up formal analysis of internet configurations. In *INFOCOM*, April 2014.

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

[43] J. Whaley. Javabdd. http://javabdd.sourceforge.net/index.html.

[44] H. Yang and S. S. Lam. Real-time verification of network properties using atomic predicates. *IEEE/ACM Trans. Netw.*, April 2016.