Formal Semantics & Verification for the Border Gateway Protocol

Konstantin Weitz Doug Woos Arvind Krishnamurthy Michael D. Ernst Zachary Tatlock
University of Washington
{weitzkon,dwoos,arvind,mernst,ztatlock}@cs.washington.edu

Abstract

Internet Service Providers (ISPs) use the Border Gateway Protocol (BGP) to exchange routing information. ISPs use a variety of formalisms, checkers, and simulators to avoid BGP configuration errors. However, these tools are based on simplified semantics of BGP or no semantics at all, and therefore they cannot guarantee the absence of router misconfigurations. Meanwhile, BGP router misconfiguration has led to worldwide outages and traffic hijacks.

To enable tools that provide formal guarantees, and to provide a foundation for future work on BGP, we present the first mechanized formal semantics of the BGP specification RFC 4271. The semantics is implemented in Coq. The semantics models all required features of the BGP specification modulo low-level details such as bit representation of update messages and TCP.

Three case studies show how to use our semantics to develop reliable proofs, checkers, and simulators; and provide evidence for the correctness of our semantics. 1) We formalized and extended the seminal pen-and-paper proof by Gao & Rexford on the convergence of BGP, revealing necessary extensions to Gao & Rexford's original assumptions. 2) We verified the soundness of the Bagpipe tool which automatically checks that BGP configurations adhere to given specifications. 3) We tested the popular BGP simulator C-BGP against our semantics, revealing a bug in C-BGP.

1. Introduction

The Internet is a collection of interconnected networks run by universities, corporations, regional ISPs, and nation-wide ISPs. These networks, collectively known as Autonomous Systems (ASes), use the Border Gateway Protocol (BGP) to exchange route announcements that describe the paths that packets (e.g. sent via TCP) can take to travel across the Internet. To route packets reliably and securely, ASes must configure their BGP routers to restrict how route announcements can be used and exchanged. For example, to avoid unprofitable routes, an AS codifies its contracts with other ASes in its BGP router configurations. Because BGP gives ASes freedom to configure their routers, BGP provides very few general guarantees — essentially all desirable properties have to be proven for a particular topology and set of router configurations.

Router configuration is a challenging and error-prone task for ASes. Large ASes maintain millions of lines of frequently-changing configurations that run distributed across hundreds of routers [23, 38]. Router misconfigurations are common and have led to highly visible failures affecting ASes and their billions of users. For example, in 2009, YouTube was inaccessible worldwide for several hours due to a misconfiguration in Pakistan [6]. In 2010 and 2014, China Telecom hijacked significant but unknown fractions of international traffic for extended periods [11, 36, 27, 25]. Goldberg surveys several additional major outages and their causes [18]. Less visible is the high cost ASes pay every day to develop and maintain configurations.

Given BGP's vital role, there exist router configuration guidelines [17, 20, 8, 40], checkers that statically check for router misconfigurations [14, 15, 4], and simulators that dynamically check for router misconfigurations [33, 35, 31, 28]. These aim to help ASes correctly configure their routers, but they fall short of providing guarantees about the absence of certain router misconfigurations, because they are based on simplified semantics of BGP or no semantics at all. For example, Gao & Rexford [17] provide configuration guidelines. These guidelines were hugely successful, as they formalized existing best practices in router configuration, provide monetary benefit to the individual ASes, and are proven to prevent Internet wide BGP divergence. Their proof however is based on a simplified BGP semantics that does not accurately model the route exchange within an AS, so their proof does not guarantee that the guidelines achieve their goal in realistic scenarios.

To improve upon this situation, this paper presents the first mechanized formal semantics of the BGP specification RFC 4271 [34]. In contrast to previous semantics [17, 20, 40], our semantics is fully formal (it is implemented in the Coq proof assistant) and models all required features of the BGP specification modulo low-level details such as bit representation of update messages and TCP.

We performed three case studies to provide evidence for the correctness of our semantics, and to show how to use our semantics as a basis for reliable proofs, checkers, and simulators that help BGP administrators avoid router misconfiguration.

1. Formalizing and extending Gao & Rexford's proof Gao & Rexford [17] proposed a set of guidelines for BGP router configuration, and they proved Internet-wide route convergence if these guidelines are implemented by every AS on the Internet.

The pen-and-paper proof by Gao & Rexford makes simplifying assumptions about the BGP protocol. For example, routers have access to all the routes received by other routers within the same AS, routes are not sent over a network but are instantly accessible whenever a router is "activated", and route announcements cannot be withdrawn; all these assumptions are frequently violated in practice.

We have extended and formalized Gao & Rexford's informal proof. Our proof is formal, mechanized, and uses our semantics of RFC 4271, which eliminates the aforementioned simplifying assumptions. Because the proof involves a more accurate model that introduces new complications, the proof requires additional insights. For example, because our semantics models both intra-domain and inter-domain routing, we have to prove intra-domain convergence of each AS. Attempting this proof revealed that the guidelines proposed by Gao & Rexford are not sufficient to prove intra-domain convergence. We thus provide an extension to Gao & Rexford's configuration guidelines, and use them to prove convergence.

This extension provides AS administrators with a guideline on how to configure intra-domain routing to avoid divergence, and provides network researchers with insights on how to accurately model intra-domain routing while simulating Internet topologies.

2. Verifying the soundness of the Bagpipe tool Bagpipe [44] defines a declarative domain-specific language that enables BGP administrators to express control-plane specifications, such as "*an AS's routers will never accept routes for invalid IP addresses*", "*an AS's routers will always forward certain routes to other ASes*", and "*an AS's routers will always prefer routes from customers over routes*

from providers". Given a specification expressed in this language, Bagpipe automatically verifies that an AS's router configurations satisfy the given specification.

Bagpipe's domain-specific language is rich enough to express specifications inferred from real AS configurations, express specifications found in the literature (such as the Gao-Rexford guidelines [17] and prefix-based filtering [29]), and express specifications for 10 configuration scenarios from the Juniper TechLibrary [24, 5].

Although Bagpipe found 19 apparent errors in three ASes with over 240,000 lines of Cisco and Juniper BGP configuration, the Bagpipe tool itself has not been verified. Therefore, its results might not be trustworthy. Using our semantics, we formally verified that Bagpipe is sound, i.e. it will never falsely claim that an AS correctly implements a specification.

3. Testing C-BGP C-BGP [33] is a popular and widely-used open-source BGP simulator. C-BGP runs a simulation of a BGP network. This simulation outputs a trace that captures all the route announcements exchanged by the routers in the BGP network, and the routes installed in each router's routing information bases. We compared C-BGP against our semantics via randomized differential testing [13].

We ran C-BGP and our semantics over 100,000 times on BGP networks with randomly-generated topology, router configurations, and initial routes. Some tests revealed that C-BGP occasionally sends announcements even when the routes they are advertising have not changed. This is not permitted by Section 9.2 of the BGP specification, and it is therefore rejected by our semantics. We reported this bug to the C-BGP maintainer, who acknowledged that the current behavior is incorrect. This bug implies that C-BGP might not provide the right measurements with respect to route convergence and thereby provide biased results.

Contributions We have defined a formal, mechanized semantics for BGP in Coq. We used it to extend and formalize a proof about the convergence of BGP, to verify a BGP checker, and to test a BGP simulator. These activities provide evidence that our semantics is correct and is useful for the development of reliable tools and guidelines that help BGP administrators avoid router misconfiguration.

This paper's contributions include:

- The first mechanized formal semantics of the BGP specification RFC 4271 [34] which is implemented in Coq. (Section 2).[1].
- A formalization and extension of the pen-and-paper proof by Gao & Rexford on the convergence of BGP, revealing necessary extensions to Gao & Rexford's original configuration guidelines (Section 3).
- A soundness proof of the Bagpipe tool that checks that BGP configurations adhere to given specifications (Section 4).
- A random differential tester for the BGP simulator C-BGP, revealing one bug (Section 5).

2. BGP Semantics

This section presents our formal semantics of BGP. It is the first formal semantics of the BGP specification RFC 4271 [34]. We used this semantics to formalize proofs about BGP (Section 3), to build and verify BGP checkers (Section 4), and to test BGP simulators (Section 5). Our semantics is formalized in 287 lines of Coq code (excluding comments).

The presentation of our BGP semantics is split into four parts. Section 2.1 describes how BGP sends messages between routers. Section 2.2 describes how BGP routers process messages. Section 2.3 describes the parameterization of our semantics. Section 2.4 compares our semantics to RFC 4271.

[1] Our semantics is open-source, available at `bagpipe.uwplse.org`

$IP := [0,256) \times [0,256) \times [0,256) \times [0,256)$ — ip addresses
$P := IP \times [0,33)$ — prefixes

$R \subseteq IP$ — routers
$ASN := uint16$ — AS number
$asn : R \to ASN$ — router's AS number

$C \subseteq R \times R$ — connections between routers
$in(r) := \{s \mid (s,r) \in C\} \cup \{injected\}$ — r's incoming connections
$out(r) := \{d \mid (r,d) \in C\}$ — r's outgoing connections
$md(s,d) = \texttt{if } asn(s) = asn(d) \texttt{ then } ibgp \texttt{ else } ebgp$ — link mode

$A = \{$ — BGP attributes
  $pref : uint32;$ — local preferences
  $communities : \mathcal{P}(uint32);$ — communities ($\mathcal{P}$ is powerset)
  $path : list(ASN);$ — AS path
  ...
$\} \cup \{na\}$ — used when there is *no available* route to a prefix

$M = P \times A$ — update message

Figure 1. General Semantics Definitions.

2.1 Network Semantics

The Internet consists of a network of routers that forward data packets toward their destination IP addresses. Routers announce routes—a path through other routers to a destination—via the Border Gateway Protocol (BGP). Routers announce routes by sending BGP *update messages* to one another. An update message means "I can forward packets to the following destinations." To a first approximation, routers send update messages in the following two cases. *Injection*: A BGP router that can directly deliver packets to a certain destination announces that route by sending an update message to each neighboring router. *Forwarding*: When a BGP router receives a route announcement via an update message, it processes the message and then possibly announces the route by sending an update message to each neighboring router. The rules governing this behavior, which is controlled by router configurations, are made more precise in Section 2.1.4.

Each router thus learns of a route either via injection or the receipt of an update message. Each router selects at most one of the learned routes per destination, and forwards packets for that destination to the router from which the selected route was received. This process continues until the packet reaches the router that knows how to directly deliver packets to the destination. A router's *control plane* selects and forwards routes. The control plane runs separately and asynchronously from the router's *data plane*, which forwards packets using the routes selected by the control plane. BGP operates on the control plane. Our semantics therefore models only the control plane, not the data plane.

2.1.1 Topology

The top part of Fig. 1 shows how our semantics represents the network topology.

The Internet's routers are operated by *Autonomous Systems* (ASes) such as universities, corporate networks, regional ISPs, and nationwide ISPs. ASes are identified by globally unique 16-bit AS numbers (ASNs).

The set of routers is represented as a set of IP addresses. Each BGP router r is connected to a set of other BGP routers called r's neighbors. BGP connections are symmetric, i.e. if r is a neighbor of r', then r' is also a neighbor of r. $in(r)$ is the set of routers from which r can receive update messages; this includes all neighbors of r plus a dummy neighbor called *injected* that is used by the router to inject new routes. $out(r)$ is the set of routers to which r can send update messages; this is simply the set of all neighbors.

$N := (C \to list(M)) \times ((r : R) \to S(r))$ — network sate
$U : Type$ — uninterpreted router state
$S(r) := \{$ — router state
$\quad adjRIBsIn : in(r) \times P \to A;$ — received messages
$\quad locRIB : P \to A;$ — selected messages
$\quad adjRIBsOut : out(r) \times P \to A;$ — sent messages
$\quad uninterpreted : P \to U$ — uninterpreted state
$\}$

Figure 2. Network State. The network state N consists of link state and router state. Link state keeps track of update messages currently in-flight $list(M)$ for each connection C, and each routers r's router state $S(r)$ keeps track of received, selected, and sent messages, as well as uninterpreted state U used to store other protocols' routing information.

The function *md* assigns a mode to every connection. Connections between routers owned by the same AS are in *ibgp* (internal BGP) mode, and connections between routers owned by different ASes are in *ebgp* (external BGP) mode.

2.1.2 Update Messages

A BGP router announces a route to another router by sending an *update message* (RFC 4271, §4). Each update message contains the set of destination IP addresses p for which the route is being announced, as well as attributes a that provide additional information about the route. The attributes are a record whose fields (RFC 4271, §5) include an AS *path* (the AS numbers of every AS traversed by the update message; this list contains, in reverse order, the ASes that will be traversed by packets sent to the destination IP addresses in p), a local preference *pref* (an integer that influences route selection), and a set of *communities* which is uninterpreted by the BGP protocol (32-bit integers used by ASes to announce additional information about a route). Our semantics is parametric over the attribute's fields, and thus can easily be extended to support additional fields as described in Section 2.3.

A new update message overrides any previous announcement with the same set of destinations. If the new update message's attributes are set to *na* (not available), the old route is withdrawn. If this new update message's attributes are available (i.e. a record), the old route is withdrawn and replaced by the new route.

Sets of IP addresses are commonly written in Classless Interdomain Routing (CIDR) notation: $ip/size$, where ip is an IP address and *size* is a number between 0 and 32. All addresses whose initial *size* bits are the same as those of ip are in the set $ip/size$; for example, *192.168.1.0* and *192.168.1.42* are in *192.168.1.0/24* but *192.168.2.0* is not. CIDR notation specifies a set of IP addresses that start with the same prefix, so a set of IP addresses is referred to as *prefix P*.

2.1.3 Network State

BGP is a stateful protocol. We refer to the state of a BGP network as the *network state*, which consists of link state Γ and router state Σ.

The *link state* keeps track of all the update messages currently in-flight in the network. BGP sends messages via a TCP connection, thus messages on a link from one router to another are ordered. The link state is modeled as a list of update messages for each connection between two routers. The message at the head of the list is the next one to be delivered. The message at the end of the list is the most recently sent.

The state at each router r is called the *router state* $S(r)$. A BGP router maintains state consisting of three tables, known as *Routing Information Bases* (RIBs) (RFC 4271, §3.2), that keep track of the update messages received, selected, and sent by the router. The Adj-RIBs-In contains all the update messages the router has received, except those that have been subsequently withdrawn. The Loc-RIB contains all update message that the router has selected to perform packet forwarding on the data plane. The Adj-RIBs-Out contains all the update messages the router has sent, except those that the router has subsequently withdrawn. A BGP router also maintains *uninterpreted state* U. Uninterpreted state consists of all the other state tracked by the router, such as state from other routing protocols like OSPF.

The router state is updated every time that a router receives a new update message. This is modeled using the *process* function, which takes a router's state, and returns the router's new state and the messages to be sent to the router's neighbors. The implementation of *process* is defined in Section 2.2.1.

Figure 2 describes how our semantics models the network state. The network state N is a tuple consisting of the link state and each router's router state. The router state $S(r)$ of router r is a record consisting of three mappings from prefixes and neighbors to attributes, and uninterpreted state. Each mapping models one of the RIBs.

The *adjRIBsIn* field of r's state $\sigma : S(r)$ models r's Adj-RIBs-In. The *adjRIBsIn* maps every incoming neighbor i (including *injected*) and prefix p (written as $adjRIBsIn(\sigma, i, p)$) to the attributes of the update message most recently received from r's neighbor i for prefix p. It suffices to only store the most recently received update message, as the BGP specification demands that previously received update messages are implicitly withdrawn with the receipt of a new update message from the same neighbor and for the same prefix. It suffices to only store an update message's attributes, as the message's prefix is uniquely determined by the prefix the attributes are stored for. If no update message has yet been received for a neighbor and prefix, e.g. whenever the router is restarted, the *adjRIBsIn* maps that neighbor and prefix to *na* (not available). In practice, the vast majority of entries in any RIB map to *na*.

The *locRIB* field of r's state $\sigma : S(r)$ models r's Loc-RIB. The *locRIB* maps every prefix p (written as $locRIB(\sigma, p)$) to the attributes of the update message selected by the router to perform packet forwarding. It suffices to store only one record of attributes per prefix, because for a given prefix, a router selects at most one route for packet forwarding. If no update message is available for a prefix, or the router chooses to select none of them, the *locRIB* maps that prefix to *na* (not available). The BGP protocol handles prefixes independently. If two prefixes overlap (one is always contained inside the other, e.g. *192.168.0.0/16* and *192.168.1.0/24*), BGP may install routes for both prefixes, and the dataplane then chooses the route with the longest prefix (here *192.168.1.0/24*) to forward a packet (e.g. with IP *192.168.1.1*).

The *adjRIBsOut* field of r's state $\sigma : S(r)$ models r's Adj-RIBs-Out. The *adjRIBsOut* maps every outgoing neighbor neighbor o and prefix p (written as $adjRIBsOut(\sigma, o, p)$) to the attributes of the update message most recently sent to r's neighbor o for prefix p. It suffices to only store the most recently sent update message, as previously sent update messages are implicitly withdrawn. If no update message has yet been sent for a neighbor and prefix, the *adjRIBsIn* maps that neighbor and prefix to *na* (not available).

The *uninterpreted* field of r's $\sigma : S(r)$ models r's uninterpreted state. A router can store different uninterpreted state for each prefix; thus, the *uninterpreted* field maps every p (written as $uninterpreted(\sigma, p)$) to an uninterpreted state U. Storing different uninterpreted state per prefix, instead of one uninterpreted state per router, simplifies updating the state as described in Section 2.2.1. Our semantics is parametric over the uninterpreted state U, but it is intended to store the routing information needed by other protocols.

With the *na* value, our semantics unifies two concepts from the BGP specification. 1) An in-flight update message withdraws an earlier update message, if its attributes are set to *na*. 2) A RIB does

$\boxed{trace(N)}$

$$\frac{\Gamma,\Sigma \rightsquigarrow \Gamma',\Sigma' \qquad trace(\Gamma',\Sigma')}{trace(\Gamma,\Sigma)}\ \text{STEP}$$

$\boxed{N \rightsquigarrow N}$

$$\frac{}{\Gamma,\Sigma \rightsquigarrow \Gamma,\Sigma}\ \text{SKIP}$$

$$\frac{\begin{array}{c}\sigma = (adjRIBsIn(\Sigma[r], injected, p) := a) \\ (p,a) \in M \qquad \Gamma',\sigma' = process(r,p,\sigma)\end{array}}{\Gamma,\Sigma \rightsquigarrow \Gamma +\!\!+ \Gamma', \Sigma[r := \sigma']}\ \text{INJ}$$

$$\frac{\begin{array}{c}(s,r) \in C \qquad \sigma = (adjRIBsIn(\Sigma[r], s, p) := a) \\ \Gamma(s,r) = (p,a) :: ms \qquad \Gamma',\sigma' = process(r,p,\sigma)\end{array}}{\Gamma,\Sigma \rightsquigarrow \Gamma[(s,r) := ms] +\!\!+ \Gamma', \Sigma[r := \sigma']}\ \text{FWD}$$

$$\frac{\begin{array}{c}\sigma = (uninterpreted(\Sigma[r], p) := u) \\ (p,u) \in P \times U \qquad \Gamma',\sigma' = process(r,p,\sigma)\end{array}}{\Gamma,\Sigma \rightsquigarrow \Gamma +\!\!+ \Gamma', \Sigma[r := \sigma']}\ \text{UPD}$$

Figure 3. Network Semantics. A *trace* is a coinductive infinite sequence of $\rightsquigarrow$ transitions. A network state consists of the in-flight messages on each connection Γ and the state at each router Σ. *process* implements a router's message forwarding logic (see Fig. 4), and returns new messages Γ' plus the router's updated state σ'. $m[k := v]$ is a notation for map/dictionary updates, defined as $(\lambda k'.\text{if } k' = k \text{ then } v \text{ else } m(k'))$. $\Gamma +\!\!+ \Gamma'$ is a notation for pointwise list append, defined as $(\lambda c.append(\Gamma(c), \Gamma'(c)))$. The notations $(adjRIBsIn(\sigma, i, p) := a)$ and $(uninterpreted(\sigma, p) := u)$ both return a copy of state σ, with *adjRIBsIn* and *uninterpreted* updated respectively.

not contain a route for a certain neighbor and prefix, if it maps to *na* for that neighbor and prefix.

2.1.4 Traces

Our semantics models an execution of the BGP protocol as a coinductive sequence of network state transitions (a trace), starting from some initial network state.

BGP routers can only perform a fixed set of actions to *transition* from some network state (Γ,Σ) to the next network state (Γ',Σ'). Our semantics models these transitions with the $(\Gamma,\Sigma) \rightsquigarrow (\Gamma',\Sigma')$ relation defined in Fig. 3. Most transitions invoke the *process* function, which implements a router's message-forwarding logic (described later in Fig. 4).

A *trace* is a sequence of all the transitions taken by an execution of BGP from some initial state. The BGP protocol might never converge—for instance, update messages might be sent in an infinite loop around a network. A trace is thus modeled as a coinductive infinite sequence of transitions.

Figure 3 describes all four possible network state transitions: skipping SKIP, injection INJ, forwarding FWD, and uninterpreted state updating UPD.

Skipping The SKIP transition marks a point in an execution of the BGP protocol when nothing happens. Neither the link state nor any of the router states are updated.

Injection Some BGP routers know how to directly deliver packets for a destination prefix p without involving any other AS. In practice, routers learn of this fact either via static configuration, or via some other network protocol such as OSPF [10]. RFC 4271 is vague in its description of how BGP routers should implement injected routes (RFC 4271, §9.4). Our semantics models it using the dummy *injected* neighbor and the INJ transition. The INJ transition marks a point in an execution of the BGP protocol when a BGP router r learns of a new route to directly deliver packets. This route consists of a prefix p and some attributes a that provide additional information about the route. The router r first installs the injected route in the *adjRIBsIn* for its dummy neighbor (*injected*), and then process the route using the *process* function which takes the router's updated state, and returns the router's new state σ' and the messages Γ' to be sent to router r's neighbors. For each connection c, the new messages for c in Γ' are appended to the existing messages for c in Γ. Note that *any* route can be injected, and any injected route can be withdrawn with *na*; *process* provides a configuration hook to filter unwanted injected routes.

Forwarding The FWD transition marks a point in an execution of the BGP protocol when a BGP router r has received a route announcement via an update message $m = (p,a)$ from some neighbor s. This is only possible if, at the beginning of the transition, the link state for the connection between s and r is a list with m as its first element. The router r first installs the received message in the *adjRIBsIn* for its neighbor s, and then processes this new route by calling the *process* function. The message m is removed from the link that it was received from. For each connection c, the new messages for c are appended to the existing messages for c in Γ.

Uninterpreted State Update The UPD transition marks a point in an execution of the BGP protocol when a BGP router r changes its uninterpreted state for prefix p. The router r then reprocesses all the routes of prefix p by calling calling the *process* function. Storing a router's uninterpreted state for each prefix, instead of having just one uninterpreted state per router, has the benefit that a router can avoid reprocessing update messages for all other prefixes, if the update will only affect processing for a subset of prefixes.

2.2 Router Semantics

This section describes how a BGP router processes routes in its Adj-RIBs-In.

2.2.1 Router Processing

The BGP specification requires a router to execute several steps in response to a change of its Adj-RIBs-In (e.g. because of an incoming message) (RFC 4271, §9) for a given prefix; the router's configuration customizes some of these steps. The steps are:

1. The configurable *import* step modifies the attributes of each received message for the given prefix, resulting in imported messages. This step can, for example, be useful to filter incoming messages with invalid prefixes.
2. The *decision* step selects a single message from all imported messages for the given prefix. This step can, for example, be useful to prefer messages from paying neighbors over messages from neighbors that expect to be paid.
3. The configurable *export* step modifies the selected attributes for the given prefix, resulting in an exported message. This message is sent to the router's neighbors. This step can, for example, be useful to block update messages to competitors.

These steps are implemented by the *process* function (Fig. 4). The function is invoked as, for example, $process(r,p,\sigma)$ whenever router r with state σ receives an update message for prefix p.

The router first imports the attributes from all neighbors in the Adj-RIBs-In σ_i and stores the result in the variable I (this corresponds to step 1 from above). Attributes are imported with a call to the *imp* function, which takes the current router r, prefix p, and neighbor i, as well as the attributes stored in σ_i for that

$imp : (r : R) \to in(r) \to P \to A \to A$ — configure import
$exp : (r : R) \to in(r) \to out(r) \to P \to A \to A$ — configure export
$dec : (r : R) \to U \to (in(r) \to A) \to in(r)$ — select message

$process : (r : R) \to P \to S(r) \to ((C \to list(M)) \times S(r))$
$process(r, p, \sigma) :=$
  let $\sigma_i := adjRIBsIn(\sigma)$
      $I := \lambda i.imp(r, i, p, \sigma_i(i, p))$
      $i^* := dec(r, uninterpreted(\sigma, p), I)$
      $a^* := I(i^*)$
      $\sigma_l := locRIB(\sigma)[p := a^*]$
      $\sigma_o := \lambda(o, p').$ if $p' = p$ then $exp(r, i^*, o, p, a^*)$
                    else $adjRIBsOut(\sigma, o, p')$
      $\Gamma := \lambda(s, d).$ if $s = r \wedge adjRIBsOut(\sigma, d, p) \neq \sigma_o(d, p)$
                  then $[(p, \sigma_o(d, p))]$ else $[]$
  in $(\Gamma, \{adjRIBsIn := \sigma_i; locRIB := \sigma_l; adjRIBsOut := \sigma_o;$
        $uninterpreted := uninterpreted(\sigma)\})$

Figure 4. Router Semantics. *process* defines how a router processes attributes stored in its Adj-RIBs-In. First, r imports all attributes from the *adjRIBsIn* σ_i. Second, r chooses the best imported attribute a^* from neighbor i^* and stores it in the *locRIB* σ_l. Third, r exports a^* to all its neighbors, storing the result in its *adjRIBsOut*. $m[k := v]$ is a notation for map/dictionary updates, defined as $(\lambda k'.\text{if } k = k' \text{ then } v \text{ else } m(k'))$.

neighbor and prefix, and returns modified attributes. The *imp* function is one of BGP's hooks for customization, defined in a router's configuration. The function is usually written in either the Juniper or Cisco configuration language, which are loop-free imperative programming languages with domain-specific syntax and semantics. Our semantics is parametric over the particular language used, and just requires that the configuration language can be denoted to a mathematical function. The advantages of this approach are described in Section 2.3. An implementation of a router can avoid re-importing old announcements by caching the result of old announcement imports. The *imp* function can discard a message by modifying the message's attributes to *na*. For example, to discard all messages for invalid prefixes *invalid*, a router can be configured with the following *imp* function: $\lambda r\, i\, p\, a.$ if $invalid(p)$ then na else a.

Next, the router uses the *dec* function to select the incoming neighbor i^* whose imported update message attributes a^* should be used for routing packets on the data plane (this corresponds to step 2 from above). How *dec* chooses a neighbor is defined in Section 2.2.2. Because *dec* depends on the router's uninterpreted state, it is necessary to reprocess the Adj-RIBs-In every time that the uninterpreted state changes. The router then stores the selected attributes in the Loc-RIB. Instead of directly updating the state, the router returns a new state which is a modified copy of the original state. The new state's *locRIB* is stored in the variable σ_l.

Next, the router exports the selected attributes to all its neighbors (this corresponds to step 3 from above). Attributes are exported with a call to the *exp* function, which takes the current router r, prefix p, and neighbor o, as well as the selected attributes a^* and the neighbor i^* from which they were selected, and returns modified attributes. The *exp* functions are configured in a similar fashion to the *imp*. The router then stores all the exported attributes in the Adj-RIBs-Out σ_o.

Finally, the router decides which update messages to send. We model the sent messages as a mapping from every connection (even those that are not connected to r) to a list of sent messages. For every connection from some source s to some destination d, the router r sends a message if and only if r is the connection's source, and the Adj-RIBs-Out σ_o for the destination neighbor d has just changed.

Our semantics models injected routes as being received from the *injected* dummy neighbor, and stores them in the *adjRIBsIn* for the *injected* neighbor. Note that *any* route can be injected into a router. The router can be configured to only select some of these routes (e.g. a small set of static routes) by dropping unwanted routes in the *imp* function for the *injected* neighbor.

2.2.2 Route Selection

The BGP specification demands that a router select the update message with the most preferable attributes, where a router r with uninterpreted state u prefers attributes a from neighbor i over attributes a' from neighbor i', iff (in order of priority) (RFC 4271, §9.1.2.2):

1. a has a higher local preference value (*pref*) than a'
2. a has a shorter AS path (*path*) than a'
3. a has a better origin value than a'
4. a has a lower med value than a'
5. i is non-internal, and i' is internal
6. r assigns a lower interior cost (e.g. computed by the OSPF protocol, and stored in u) to sending traffic to i instead of i'
7. i has a lower router identifier value than i'
8. i has a lower router address than i'

Attributes with the *na* value have strictly lower preference than any other attributes. The reader need not understand all the details of this list, as router implementations and extensions of BGP often implement a customized version of this preference relation, e.g. they may insert an additional check or change the order of the checks. For this reason, our semantics is parametric over the exact implementation of the preference relation. The advantages of a parameterized preference relation are described in Section 2.3.

Below, we first describe the preference relation $(i,a) \succeq_u (i',a')$ that our semantics is parametric over. It compares attributes a and a' received from the neighbors i and i' by a router with uninterpreted state u. Second, we describe the function *dec* which selects attributes according to this preference relation. Third, we describe a preference relation $a \geq a'$ that compares attributes a and a', without the need to know the neighbors from which they were received, or the uninterpreted state. The details of this section are crucial for the correctness of the Internet convergence theorem proven in Section 3.

Preference Relation for Incoming Attributes Our semantics is parametric over the exact implementation of the preference relation $(i,a) \succeq_u (i',a')$, it just requires that for any router r with uninterpreted state u, the $\preceq_u$ relation is a total preorder, i.e. it is reflexive ($(i,a) \preceq_u (i,a)$), transitive ($(i,a) \preceq_u (i',a') \to (i',a') \preceq_u (i'',a'') \to (i,a) \preceq_u (i'',a'')$), and total ($(i,a) \preceq_u (i',a') \vee (i',a') \preceq_u (i,a)$). The preference relation demanded by the BGP specification is in fact a total preorder.

Further, our semantics requires that for any router r with uninterpreted state u the preference relation is partially antisymmetric. If attributes a from neighbor i and attributes a' from neighbor i' are equally preferred, then their neighbors must be the same, i.e. $(i,a) \approx_u (i',a') \to i = i'$. In other words, attributes received from different neighbors are never tied on preference (they are totally ordered), i.e. $i \neq i' \to ((i,a) \prec_s (i',a') \vee (i',a') \prec_s (i,a))$. We say that attributes a from neighbor i and attributes a' from neighbor i' are equally preferred $(i,a) \approx_u (i',a')$ iff $(i,a) \preceq_u (i',a') \wedge (i',a') \preceq_u (i,a)$. Because each neighbor has a unique router address, the preference relation demanded by the BGP specification is in fact partially antisymmetric. However, it is not antisymmetric. For example, consider two announcements a and a' that differ in their communities, and were received from the same neighbor i. They will have the same preference, but will not be equal. Thus, $(i,a) \approx_u (i,a') \not\to (i,a) = (i,a')$.

Incoming Neighbor Selection Our semantics of BGP selects the route to forward packets using the $dec(r,u,I)$ function. This

function chooses the neighbor with the most preferable attributes, given a mapping I from every neighbor of router r to attributes, and the router's uninterpreted state u i.e. $\forall i, I(dec(r,u,I)) \succeq_u I(i)$. The selected neighbor is unique, because attributes from different neighbors cannot have the same preference (partial antisymmetry). The selected neighbor exists, because there is at least one neighbor (*injected*), and because $\preceq_u$ is total.

While the *dec* function cannot be configured directly, it can be configured indirectly by configuring the *imp* function to appropriately change an update message's *pref* field. For example, a router can be configured to select the cheapest available route to forward packets (ASes usually receive or pay money whenever they send a packet to another AS), by configuring the *imp* function to set a higher *pref* value for cheaper routes.

Preference Relation for Attributes The preference relation $\preceq_u$ defines how to compare attributes at the same router, but it is sometimes necessary to also compare attributes at different routers with different uninterpreted states. Fortunately, the checks 1-4 performed by $(i,a) \preceq_u (i,a')$, up to (but not including) the non-internal vs internal check, only depend on the attributes a and a', not the routers i, i', and the uninterpreted state u. We can thus say that attributes a are preferred over attributes a' ($a \geq a'$), if a is preferred over a' up to the non-internal vs internal check. The *na* value has strictly lower preference than any other attributes. This relation is a total preorder, but it is not antisymmetric and thus not an order (i.e. two different attributes may be equally preferred). We say that attributes a and attributes a' are equally preferred ($a \simeq a'$) iff $a \leq a' \wedge a' \leq a$.

Because attribute preference is derived from the preference demanded by the BGP specification, if an announcement a from neighbor i is preferred over announcement a' from neighbor i', then a is preferred over a', i.e. $(i,a) \succeq_u (i',a') \rightarrow a \geq a'$ (the converse is not necessarily true).

Further, because this preference relation performs all checks up to the non-internal vs internal check, if an announcement a_e is preferred over an announcement a_i, and given a non-internal neighbor i_e and an internal neighbor i_i at some router r with uninterpreted state u, then a_e received from i_e is preferred over a_i received from i_i, i.e. $a_e \geq a_i \rightarrow (i_e,a_e) \succeq_u (i_i,a_i)$.

2.2.3 Configuration Restrictions

BGP's configurability is one of the key reasons for its success, but to aid reasoning about the correctness of BGP configurations, the BGP specification places some restrictions on the *imp* and *exp* functions. Our formalizations of these restrictions are shown in Fig. 5.

Restriction 1 and 2 models that *imp* and *exp* functions may not create attributes "out of thin air" for update messages that are *na*. Restriction 3 models that *exp* functions must forward update messages at most once within an AS. This avoids routing loops. Restriction 4 models that *imp* functions must drop update messages with loops in their path. Restriction 5 models that *exp* functions must extend an update message's AS path whenever a message crosses an AS border.[2]

Because real-world BGP configuration languages often enforce a subset/superset of the above restrictions, our semantics was developed to make it easy to enforce only a subset/superset of the restrictions. For example, the router configuration language used by Juniper-manufactured routers does not enforce restriction 5, and allows arbitrary manipulations of the AS path. On the other hand, the C-BGP router configuration language[33], unlike the specification, ensures that update messages may not be sent back along the connection that they came from.

[2] The BGP spec allows $asn(r)$ to be repeated multiple times to influence tie-breaking.

1. $\forall r\, i\, p.\, imp(r,i,p,na) = na$
2. $\forall r\, i\, o\, p.\, exp(r,i,o,p,na) = na$
3. $\forall r\, i\, p\, a.\, md(i,r) = md(r,o) = ibgp \rightarrow exp(r,i,o,p,a) = na$
4. $\forall r\, i\, p\, a.\, asn(r) \in path(a) \rightarrow imp(r,i,p,a) = na$
5. $\forall r\, i\, o\, p\, a.\, md(r,o) = ebgp \rightarrow$
   $path(exp(r,i,o,p,a)) = asn(r) :: path(a)$

Figure 5. Rule Restrictions. BGP requires that the *imp* and *exp* rules cannot create attributes "out of thin air" (1,2), avoid forwarding loops (3,4), and extend paths appropriately (5).

2.3 Parameterization

Our BGP semantics is parametric over attributes (Section 2.1.2), attribute preference (Section 2.2.2), uninterpreted state (Section 2.1.3), and router configuration languages (Section 2.2.1).

The fact that our semantics is parameterized has several advantages: 1) it *simplifies* the semantics' definition (e.g. we did not have to formally model all the languages used to configure routers in practice); 2) it enables *extending* the semantics (e.g. RFC 1997[7] adds an additional attributes field, and RFC 4456[3] changes the attribute selection algorithm); and 3) it enables reasoning about *non-compliant* router implementations (e.g. actual router implementations have their own attribute selection algorithm).

While parameterization does simplify the semantics, we do *not* loose any guarantees. A proof that holds for all possible parameter instantiations of our semantics, also holds for the one actually used by the verified BGP network.

2.4 Comparison to RFC 4271

Our semantics models the full BGP specification (RFC 4271) except for low-level details (bit representation of update messages, version negotiation, establishing BGP connections, and TCP). It models some extensions such as the communities attribute and route reflectors. It does not model all optional features, such as route aggregation. We believe that our semantics could be extended with these additional features.

The BGP protocol exhibits some surprising behavior that is nevertheless correctly modeled by our semantics. Following are two such cases.

A BGP router treats all prefixes independently, even if they overlap. If a router receives two routes for overlapping prefixes, e.g. one for prefix $p = 192.168.0.0/16$ and one for prefix $p' = 192.168.1.0/24$, r will install both routes in its Loc-RIB. Whenever a packet needs to be forwarded on the data plane, e.g. with IP *192.168.1.1*, the router will choose the route with the longest prefix (in this case p'), even if the attributes of the route for the other prefix are more preferable according to *dec*.

A BGP router may explicitly withdraw an available route. Consider a router r that has imported attributes a from its neighbor i, has installed a in its Loc-RIB, and has exported a to its neighbor o. If r subsequently imports more preferable attributes a' from some other neighbor i', r will install a' in its Loc-RIB, and export a' to neighbor o. If a' is successfully exported, a will be withdrawn implicitly; if a' is dropped on export, a will be withdrawn explicitly. In the second case, the available route a will be withdrawn from neighbor o without replacement.

3. Proof of Gao & Rexford's Guidelines

This section describes our formalization (in the Coq proof assistant) of Gao & Rexford's router configuration guidelines and of their proof that any BGP network implementing these guidelines will eventually converge.

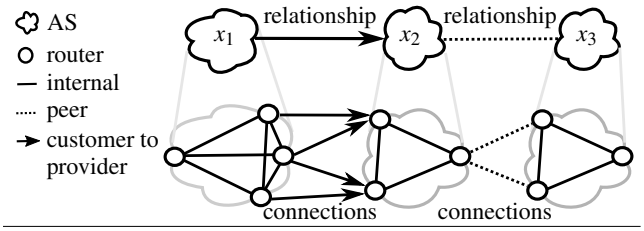

Figure 6. AS Router Endomorphism. Every AS has at least one router. The connections between routers of any two ASes, match the relationship between the ASes.

Unlike most other networking protocols, BGP has no built-in mechanism to guarantee network convergence (i.e. routers may indefinitely flip-flop between routes). Network convergence is however vital, as a divergent BGP network may not properly forward packets, and will waste networking resources. A large number of papers have been published to address the problem of BGP divergence [20, 8, 42]. Gao & Rexford's guidelines were also hugely successful in practice, as they formalized existing best practices in router configuration, provide monetary benefit to the individual ASes, and are proven to prevent Internet-wide BGP divergence.

This section is split into three subsections. Section 3.1 formalizes Gao & Rexford's configuration guidelines, as well as their assumptions about the BGP network's topology. We also present an extension to Gao & Rexford's original guidelines to prevent internal AS divergence. Section 3.2 presents our formal proof that any BGP network implementing these guidelines will eventually converge. This proof follows Gao & Rexford's original argument, but extending it to our semantics of RFC 4271 requires additional insights. Section 3.3 compares our proof with the one provided by Gao & Rexford.

3.1 Assumptions and Configuration Guidelines

This section formalizes Gao & Rexford's configuration guidelines, as well as their assumptions about the BGP network's topology. Unless otherwise noted, any assumption made by our proof is also an assumption made by Gao & Rexford's original proof.

3.1.1 Topology Restrictions

ASes have various business relationships with one another. This proof assumes (just like the proof by Gao & Rexford) that the relationship between any two ASes x and x' is exactly one of the following three: *Customer to Provider*: the customer AS x pays the provider AS x' for every packet sent between the two (in either direction). Customers are usually smaller ASes that pay the larger provider AS to get access to the entire Internet. We also say that x' is in a provider-to-customer relationship with x. *Peering*: the peer AS x neither pays nor charges the peer AS x' for any packet sent between the two (in either direction). The peering relationship is symmetric. *No Relationship*: the two ASes do not directly send packets from one to the other.

This proof assumes that the connections between routers of any two ASes match the relationship between the ASes. This is illustrated in Fig. 6. Formally, there is a endomorphism between the ASes with their relations, and routers with their connections. In other words: every router belongs to exactly one AS, and every AS has at least one router; any two ASes that are in a customer to provider relationship only maintain customer to provider connections between their routers, and there exists at least one such connection; any two ASes that are in a peering relationship only maintain peering connections between their routers, and there exists at least one such connection; any two ASes that are not in a relationship maintain no connections between their routers.

This proof assumes that each AS is in a full-mesh configuration: each router is directly connected to all other routers of the same AS. This simplifies the proof of convergence, but in practice some large ASes avoid the performance penalty of a full-mesh configuration by using route reflectors [3]. Route reflectors are an optional RFC extension, supported by our semantics but not by this proof.

This proof assumes that the customer to provider relationships form a Directed Acyclic Graph (DAG). See Fig. 10 for an example. This means that each AS can be assigned a level (its *AS level*), such that any AS is only a provider to lower-level ASes, and only a customer to higher-level ASes. This assumption is realistic. ASes on the lowest level are usually universities or companies that require a provider to forward and receive packets from the rest of the Internet, but are not themselves providers to any other ASes. On the second lowest level are Internet Service Providers like Internet2, which is a provider to many universities in the US, but is not large enough to route all prefixes without talking to a provider. On the highest level are ASes like (the cleverly named) Level3. These can route any prefix without talking to any provider. Our proof even holds for DAGs consisting of multiple disconnected components, though this is not usually the case for the Internet.

Lastly, this proof assumes that the customer to provider relationships are well-founded, i.e. every customer's direct or indirect provider can be reached via a finite number of ASes. Similarly, the provider to customer relationships must be well-founded.

3.1.2 Network Assumptions

This proof assumes that update messages sent by a BGP router are eventually delivered. This property is referred to as *fairness*. Formally, this proof assumes that for every link state in a trace, every non-empty connection is associated with a natural number n such that after n transitions in the trace, the trace contains a FWD transition that delivers the update message on that link.

This proof further assumes that there exists exactly one AS that knows how to directly route a certain prefix. Formally, for every prefix p, there exists exactly one AS $x_0(p)$ with exactly one router $r_0(p)$ that injects the route for that prefix with attributes $a_0(p)$. This is usually the case on the Internet, but ASes may violate this assumption due to misconfiguration [6].

This proof further assumes that after an initial phase where injections can happen, no new injections will happen, and no existing injections will be withdrawn (i.e. the trace will contain no INJ transitions). This is an unrealistic assumption. The Internet as a whole frequently injects new routes and withdraws already injected routes, such that the Internet is never in a truly stable state. However, because the BGP protocol handles prefixes independently, there is usually enough time for any particular prefix to become stable, even if some other prefixes are still converging.

Finally, the proof assumes that none of the routers will ever update their uninterpreted state (i.e. the trace will contain no UPD transitions). This is again an unrealistic assumption, but works in practice because only few routers have to choose between routes based on a router's uninterpreted state.

3.1.3 Configuration Guidelines

Gao & Rexford's proof requires that every AS x configures all its routers to follow three guidelines.

Import Guideline: Routes from customers are preferred over routes from peers or providers. Formally, attributes imported by a router r from a customer i_c must have strictly higher preference than attributes a' imported by any router r', of the same AS, from a peer or provider i_p, i.e. $imp(r,i_c,p,a) > imp(r',i_p,p,a')$ ($>$ is defined in Section 2.2.2). This guideline not only aids the Internet's global convergence, it also makes financial sense to the individual AS. By preferring update messages from customers, an AS's routers will

prefer to forward packets to paying customers, leading to increased revenue for the AS.

Export Guideline: Only routes involving customers are exported. Formally, a router r of x may only export update messages m that satisfy one of the following three conditions: m is being forwarded to a customer, or m was directly or indirectly received from a customer of x, or m was injected. Router r received a message from a customer indirectly, if some router r' of x received the message from a customer and subsequently forwarded it inside the AS to the router r. This guideline not only aids the Internet's global convergence, it also makes financial sense to the individual AS. By blocking update messages that do not involve customers, the AS avoids establishing routes that do not involve customers, and thus avoids having to forward packets that neither the sender nor receiver is willing to pay for.

Injection Guideline: Injected routes are preferred over all other routes. Formally, AS x_0's injected attributes a_0 must have strictly higher preference than attributes a imported by any router r, of the same AS, from any external neighbor i for the same prefix p, i.e. $imp(r_0, injected, p, a_0) > imp(r, i, p, a)$. By preferring injected routes, and exporting them to all neighbors, an AS makes the route's prefix available outside the AS, which is usually the intended behavior.

Our proof extends the above guidelines by Gao & Rexford with the following guideline:

Internal Guideline: A router is required not to change the preference of attributes imported or exported internally. Formally, for any internal neighbor n of r, $imp(r, n, p, a) \simeq a$ and $exp(r, i, n, p, a) \simeq a$ ($\simeq$ is defined in Section 2.2.2). This guideline extension is crucial for our proof (see Section 3.2.1), but was overlooked by Gao & Rexford's original proof due to their simplified semantics of BGP. Individual ASes have financial incentives to follow this guideline, as it prevents internal divergence, and thus prevents unnecessary internal traffic that the AS would have to provision resources for. Many ASes, for example Internet2, already follow this guideline.

3.2 Convergence Proof

The goal of this section is to show that any BGP network will converge, assuming it follows the assumptions and guidelines from the previous Section 3.1.

A BGP network *converges*, iff for every execution trace of the BGP protocol from some valid initial state, the BGP network will eventually become stable.

Some proposition P happens *eventually*, iff there exists a natural number n, such that P holds for every network state of the trace after the first n transitions. This terminology is inspired by modal logic.

A BGP network will eventually become *stable*, iff every AS of the BGP network will eventually become stable for every prefix. An AS will eventually become stable for a prefix p, iff every router of the AS will eventually become stable for prefix p. A router r will eventually become *stable* for prefix p, iff it will eventually select some attributes for prefix p that will never change. Formally, there exist best attributes a_r and a best incoming neighbor i_r, such that: 1) r's Adj-RIBs-In for p and i_r^* will eventually contain a_r, and 2) the imported a_r will eventually be preferred over any other attributes a for p imported from any other incoming neighbor i of r. r's Loc-RIB will thus eventually contain the imported attributes a_r, and all of the router's outgoing connections will eventually become stable for p.

The connection from one router s to another router d will eventually become stable for prefix p, iff the list of messages for prefix p on the connection will eventually be empty. Once stable, d's Adj-RIBs-In for s will contain the attributes from s's Adj-RIBs-Out for d.

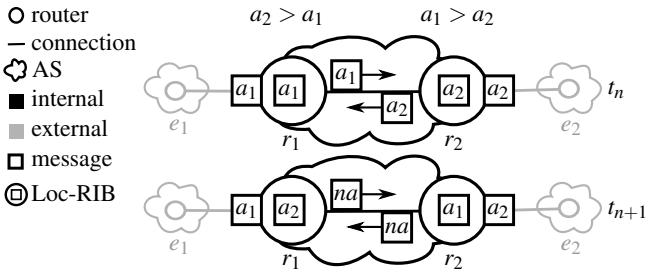

Figure 7. Divergence Example. The AS with router r_1 and r_2 can transition between the network state at t_n and t_{n+1} forever, and is thus not guaranteed to eventually become stable.

An initial state is *valid*, iff all RIBs map to *na*, and there are no update messages on any connection; except that for any prefix p the injecting router $r_0(p)$ has injected its announcement $a_0(p)$.

Going forward, we prove all statements of stability for some given BGP network that complies with Section 3.1, some given execution trace of that network, and some given prefix p. Our final theorem will combine these proofs into a proof that holds for *all* compliant BGP networks, traces, and prefixes.

The proof that a BGP network will converge proceeds in two steps. Section 3.2.1 proves that a single AS will eventually become stable, assuming that some of the incoming connections to the AS will eventually become stable. This proof is novel. Section 3.2.2 uses the local convergence proof to show that a BGP network converges globally. This proof follows Gao & Rexford's proof, but required some novel insights.

3.2.1 Internal Stability

The goal of this section is to show that an AS will eventually become stable, assuming that some of the incoming connections to the AS will eventually become stable. Note that Gao & Rexford's simplified semantics of BGP does not accurately model interactions between routers within a single AS, and they did not prove any results regarding internal stability.

To motivate our guideline extensions, consider the following proof that even without external update messages, an AS may fail to converge internally.

Example (Internal Divergence). *Without our import guideline, there exists an AS that is not guaranteed to eventually become stable, even assuming that* all *incoming connections to the AS will eventually become stable.*

Proof. Consider the example from Fig. 7. The AS x consists of two routers r_1 and r_2. To show that x is not guaranteed to eventually become stable, we construct a trace in which x's routers keep changing their Loc-RIBs forever.

Let the network topology surrounding e_1 and e_2, be such that we can construct a partial trace where e_1 and e_2 will eventually be stable, and that this trace leads to a network state where r_1 has received a message with attributes a_1 from e_1, and r_2 has received a message with attributes a_2 from e_2. The AS's incoming connections will thus eventually become stable.

Now consider the case where router r_1 prefers the attributes a_2 over a_1, and router r_2 prefers the attributes a_1 over a_2. This can for example be implemented by increasing the *pref* value in the *imp* function for messages received from internal routers (which violates our internal import guideline).

Now consider the following extension of the partial trace. Each router will select the attributes received from the external neighbor,

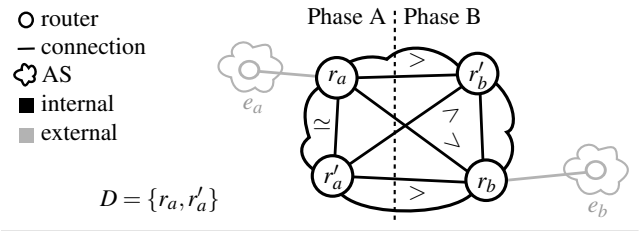

Figure 8. Internal Convergence Example. The routers r_a and r'_a in D dominate (their routes are better than) the AS's other routers. The AS will therefore eventually become stable.

and will export those attributes to the other internal router. This is the situation at time t_n in Fig. 7.

Next, consider extending the trace such that the routers receive the attributes just sent over the internal connection. Router r_1 receives a_2. r_1 prefers a_2 over a_1 and selects it. Further, r_1 exports a_2 to the internal neighbor r_2. *exp* drops a_2 as demanded by rule 3 in Fig. 5. *na* is not equal to a_1, which was previously stored in the Adj-RIBs-Out, and r_1 thus withdraws the previously sent attributes a_1 from router r_2 by sending *na*. The same happens at r_2 except with different attributes. This is the situation at time t_{n+1}.

Next, consider extending the trace such that the routers receive the withdraws. Now that the internal message has been withdrawn, they select the external message and do the appropriate export. This is the same situation as the one at time t_n. We can thus continue extending the trace indefinitely, in such a way that the AS will never become stable. □

Gao & Rexford overlooked the problem of internal stability, because the original proof by Gao & Rexford uses a simplified semantics of BGP where every router has instant access to all messages received by all other routers of the same AS, and an AS thus instantaneously becomes stable, once all incoming connections have become stable.

The goal henceforth is to show that an AS x implementing our guideline extension will eventually become stable, assuming that there exists a non-empty subset D of x's routers such that: 1) every router r in D will eventually become non-internally stable, and 2) D is dominant (non-internally stable and dominant are defined below).

A router will eventually become *non-internally stable* iff it will eventually be stable for *non-internal* (i.e. either injected or external) incoming neighbors. Formally, there exist *best non-internal attributes* a_r and a *best non-internal incoming neighbor* i_r, such that: 1) r's Adj-RIBs-In for i_r^* will eventually contain a_r, and 2) the imported a_r will eventually be preferred over any other attributes a imported from any other non-internal incoming neighbor i of r. Once non-internally stable, the router's Loc-RIB might still change.

A set of non-internally stable routers D is *dominant*, iff the imported best non-internal attributes of all routers in D are equally preferred, and the imported best non-internal attributes are strictly more preferable than any imported non-internal attributes of any other router in the AS. This is illustrated in Fig. 8. Formally, for any two routers r and r' of the AS x where r is in the set D, the following must hold: 1) If r' is in D, then r's imported best non-internal attributes a_r^* must have the same preference as r''s imported best non-internal attributes $a_{r'}^*$, i.e. $a_r^* \simeq a_{r'}^*$. 2) If r' is not in D, then r's imported best non-internal attributes a_r^* must always be strictly preferred over any imported non-internal attributes a' of r', i.e. $a_r^* > a'$.

The proof that the AS will eventually become stable proceeds in three steps. The *Internal Link Invariant* shows that the attributes of any internally sent messages have equal or worse preference than the best non-internal attributes. *Claim A* shows that routers in D will eventually become stable. *Claim B* shows that all other routers of x will also eventually become stable.

Lemma (Internal Link Invariant). *Eventually, the attributes of any update message on any internal link will have equal or worse preference than the best non-internal attributes.*

Proof. Any update messages that are already on internal connections will eventually be delivered due to fairness. It thus suffices to only show that new update messages sent via internal connections have equal or worse preference than the best non-internal attributes. We have to consider two cases: 1) A router (e.g. router r_a in Fig. 8) exports an update message received from an internal neighbor (e.g. router r'_a). By rule 3 of Fig. 5, update messages from internal routers are blocked on export to other internal routers. Thus, the router may only send a withdraw (*na*), and *na*'s preference is less than or equal to any attributes (see Section 2.2.2). 2) A router exports an update message received from a non-internal neighbor (e.g. router e_a). Because of dominance, the attributes of any update message from a non-internal neighbor are less than or equal to the best non-internal attributes. The same will be the case for the exported message, due to our extension to Gao & Rexford's guidelines (see Section 3.1.3). □

Lemma (Claim A). *Any router r in D (e.g. router r_a) will eventually become stable.*

Proof. By the definition of dominance, we already know that r's non-internal incoming attributes will eventually become stable. What remains to be shown is that eventually, the router will never receive an update message with attributes a from an internal connection i, such that (i,a) is preferable over the router's best non-internal attributes (i_r^*, a_r^*).

We know that the preference of a is equal or lower to the preference of a_r^* by the Internal Link Invariant. Even if a's preference is equal to a_r^*'s preference, a_r^* will be preferred because update messages from non-internal neighbors are preferred over update messages from internal neighbors (see Section 2.2.2). □

Lemma (Claim B). *Any router r not in D (e.g. router r_b) will eventually become stable.*

Proof. Every routers r' in D (e.g. router r_a) will eventually become stable by Claim A. In the process, r' will export its best attributes to each of its neighbors, including router r. The attributes of the update message that is eventually received by r have equal preference to r''s best attributes by our extension to Gao & Rexford's guidelines.

Eventually, r will have received the best attributes from each dominant router. Router r is guaranteed to prefer one of these over all other attributes, by the definition of dominance, and the preference relation's partial antisymmetry (see Section 2.2.2). Once this route is selected, router r will be stable. □

Theorem (Internal Stability). *An AS will eventually become stable, assuming that a non-empty subset D of the AS's routers will eventually become non-internally stable, and the set of routers D is dominant.*

Proof. By Claim A and Claim B. □

3.2.2 Global Convergence

The proof of global convergence proceeds in three steps. The *Path Invariant* shows that any update message will first steadily increase in its AS level, and then steadily decrease in its AS level. Gao & Rexford assumed this invariant, but neither stated nor proved it explicitly. *Claim 1* shows that *Phase 1 ASes* ((in)direct providers of the injecting AS x_0) will eventually become stable. *Claim 2* shows

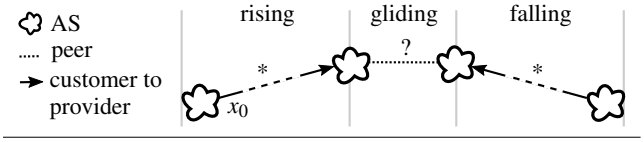

Figure 9. Path Invariant. Any update message first traverses any (indicated by $*$) number of (including 0) customer-to-provider connections, then an optional (indicated by ?) peer connection, and finally any number of provider-to-customer connections.

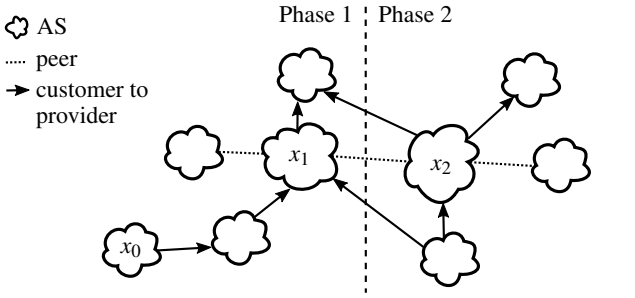

Figure 10. Global Stability Example. Illustrates a BGP network where AS x_1 is connected via all possible relationships (customer-to-provider, provider-to-customer, and peer) to various neighboring ASes from all Phases (Phase 1 and Phase 2). The same is true of x_2. Customer-to-provider relationships from Phase 1 to Phase 2 are impossible.

that *Phase 2 ASes* (all other ASes) will also eventually become stable.

Lemma (Path Invariant). *Any in-flight update message on a connection between AS x and x' is in one of three stages (as illustrated by Fig. 9):* Rising*: the message's AS level has so far strictly increased with every forwarding. This means that x is an (in)direct provider of x_0, and x' is a provider of x.* Gliding*: after the message's AS level strictly increased with every forwarding, it is now sent to a peer. This means that x is an (in)direct provider of x_0, and x' is a peer of x.* Falling*: the message's AS level has previously increased, potentially glided, and is now decreasing. This means that x is a provider of x'.*

Proof. This invariant is a consequence of Gao & Rexford's export guideline. A rising message (i.e. a message from a customer) can transition to the rising, gliding, or falling stage (i.e. be forwarded to a provider, peer, or customer); a gliding message (i.e. a message received from a peer) can only transition to the falling stage (i.e. be forwarded to a customer); and a falling message (i.e. a message received from a provider) can only transition to the falling stage (i.e. be forwarded to a customer). □

Lemma (Claim 1). *Every Phase 1 AS (an (in)direct provider of the injecting AS x_0) will eventually become stable.*

Proof. Using well founded induction on the customer-to-provider relationship, we have to show that any Phase 1 AS x will eventually become stable, assuming that every Phase 1 customer AS of x will eventually become stable. We have to consider two cases:

If x is x_0, then r_0 is non-internally stable, and the singleton set containing only the router r_0 is dominant, because a_0 is better than any attributes any router could receive from any non-internal source, by the Injection Guideline. x thus eventually becomes stable by the Internal Stability theorem.

If x is not x_0 (e.g. x is the AS x_1 in Fig. 10), then we first show that the connection from every customer to x will eventually become stable. We have to consider two cases: 1) If the customer is a Phase 1 AS, the connection will eventually be stable by the induction hypothesis. 2) If the customer is a Phase 2 AS, the connection will always be empty (and thus stable) by the Path Invariant — Phase 2 ASes are not (in)direct providers of x_0, any message sent by a Phase 2 AS must thus be in the falling stage, and any connections to non-customers are thus empty. Note that Phase 2 ASes never send any messages to Phase 1 ASes.

Knowing that the connection from every customer of x will eventually become stable, pick the set of customer connections C that have sent the best attributes. These attributes are preferred over attributes from peers or providers, because of the import guidelines. Each router connected to any customers in C is thus non-internally stable, and the set of routers connected to any customers in C is dominant. x thus eventually becomes stable by the Internal Stability theorem. □

Lemma (Claim 2). *Every Phase 2 AS (not a Phase 1 AS) will eventually become stable.*

Proof. Using well founded induction on the provider-to-customer relationship, we have to show that any Phase 2 AS x (e.g. x_2 from Fig. 10) will eventually become stable, assuming that every Phase 2 provider AS of x will eventually become stable.

It suffices to show that every router of x eventually becomes non-internally stable. If this is the case, the set of routers with the most preferred best non-internal attributes are dominant, and x thus eventually becomes stable by the Internal Stability theorem.

A router will eventually become non-internally stable, if every external incoming connection will eventually become stable.

Each external incoming connection is either from Phase 1 or Phase 2. Every connection from Phase 1 will eventually become stable by Claim 1. What remains are connections from Phase 2. There are two cases. 1) Connections from Phase 2 providers will eventually become stable by the induction hypothesis. 2) Connections from Phase 2 customers and peers will always be empty (and thus stable) by the Path Invariant — Phase 2 ASes are not (in)direct providers of x_0, any message sent by a Phase 2 AS must thus be in the falling stage, and any connections to non-customers are thus empty. □

Theorem (Global Convergence). *All guideline-conforming BGP networks converge.*

Proof. All ASes will eventually become stable for any given trace and prefix p by Claim 1 and Claim 2. As the set of prefixes is finite, this implies that the BGP network will eventually become stable for any trace. This implies that for all traces, the BGP network will eventually become stable, and thus that the BGP network converges. □

We have formally stated and verified the above theorem, along with all its assumptions and lemmas, in Coq.

3.3 Comparison to Gao & Rexford's Original Proof

The pen-and-paper proof by Gao & Rexford makes various simplifying assumptions about the BGP protocol. For example, routers have access to all the routes received by other routers within the same AS, routes are not sent over a network but are instantly accessible whenever a router is "activated", and route announcements cannot be withdrawn.

Our proof uses our semantics of RFC 4271, which eliminates the aforementioned simplifying assumptions. Because the proof

involves a more accurate model that introduces new complications, the proof requires additional insights.

Most importantly, because our semantics models both intra-domain and inter-domain routing, we have to prove local stability for each AS (Section 3.2.1), which also requires an extension to Gao & Rexford's original configuration guidelines (Section 3.1.3).

Our formal proof also extends Gao & Rexford's original proof with novel arguments about edge-cases (e.g. about the injection, availability, and withdraw of update messages), and with novel invariants about BGP (e.g. if a property P eventually hold for the update message in a router r's Adj-RIBs-Out to some neighbor r', then P will eventually hold for every message on the connection between r and r', and P will eventually hold for the update message in r''s Adj-RIBs-In for r). These edge-cases and invariants are too plentiful for an in-depth description in this paper.

Furthermore, because our proof is expressed formally in the Coq proof assistant, it also requires more rigorous and detailed reasoning. For example, the Path Invariant shows that any update message will first steadily increase in its AS level, and then steadily decrease in its AS level. Gao & Rexford assumed this invariant, but neither stated nor proved it explicitly.

Our proof totals 5123 lines of Coq code. 432 lines formalize the proof's assumptions (Section 3.1), 218 lines prove global convergence (Section 3.2.2), 1410 lines prove internal stability (Section 3.2.1), and 3063 lines prove invariants about BGP (including the Path Invariant).

4. Bagpipe

Given BGP's vital role, there exist several checkers that statically check for router misconfigurations [14, 15, 4]. These aim to help ASes correctly configure their routers, but fall short of providing guarantees about the absence of certain router misconfigurations, because they are based on simplified semantics of BGP or no semantics at all.

Bagpipe is one such checker. Bagpipe defines a declarative domain-specific language that enables an AS's BGP administrators to express control-plane specifications, such as "*the AS's routers will never select routes for invalid IP prefixes*", "*the AS's routers will always forward certain routes to other ASes*", and "*the AS's routers will always prefer routes from customers over routes from providers*". Provided with a specification expressed in this language, Bagpipe automatically verifies that an AS's router configurations satisfy the given specification.

Bagpipe's domain-specific language is rich enough to express specifications inferred from real AS configurations, express specifications found in the literature (such as the Gao-Rexford guidelines [17] and prefix-based filtering [29]), and express specifications for 10 configuration scenarios from the Juniper TechLibrary [24, 5].

Bagpipe was initially developed without any formal semantics of BGP, which made it difficult to know exactly what the tool was checking, and if it did so correctly. To evaluate the usefulness of our semantics, we formally verified the Bagpipe checker, so that it provides rigorous guarantees about the properties that it checks. Formally verifying Bagpipe using our semantics had two primary benefits: 1) it deepened our understanding of the tool, e.g. it lead to the discovery of the *initial network reduction*, which is the proof that justifies Bagpipe's verification algorithm; and 2) it ensured that Bagpipe correctly handles all the details of RFC 4271.

The Bagpipe checker along with a pen-and-paper proof of its soundness is described the Bagpipe paper [44], this section describes the formal verification of Bagpipe.

4.1 Initial Network Reduction

Given a specification and router configurations for a single AS, Bagpipe either guarantees that the specification holds, or provides a counter example to the specification's assertion. For example, a specification may assert that "*the AS's routers will never select routes for invalid IP prefixes*". If Bagpipe returns that this specification holds, it provides the guarantee that:

$$\forall t\ r\ p\ n, \texttt{let}\ \ (\Gamma,\Sigma) := t[n]\ \ \texttt{in}\ \ \mathit{invalid}(p) \rightarrow \mathit{locRIB}(\Sigma(r), p) = \mathit{na}$$

This means that for all traces t, routers r, and invalid prefixes p, the router state Σ after n transition of the trace $t[n]$, does not have attributes selected for prefix p (Bagpipe currently assumes that traces are free of UPD and INJ transitions for routers of the AS under consideration).

Because the set of all traces is infinite, Bagpipe cannot verify such a specification by searching the set of all traces for a counter example. To address this challenge, Bagpipe only searches a finite set of short trace prefixes from the initial state for counter examples, instead of the infinite space of all traces.

Before the verification process, we only had a vague intuitive understanding of why this finite search was reasonable. During the verification process, we developed the initial network reduction, which justifies searching only this finite space.

The initial network reduction formalizes the observation that if a BGP router will ever select or forward a particular update message, it would also do so immediately after initialization, i.e. before it has received any other update messages. This is because, in the initial network, the message does not have to compete with any other messages during the selection phase. A BGP network thus exhibits "maximal behavior" with respect to a given announcement at the beginning of its execution trace. Bagpipe exploits this insight to soundly verify a specification by searching for counter examples only the finite set of trace prefixes starting in the initial network. We formally verified the initial network reduction in Coq.

4.2 Bagpipe Correctly Handles RFC 4271

During our verification of Bagpipe we identified two bugs related to RFC 4271 details. Our prototype checker (1) did not always correctly verify specifications for *na* attributes, and (2) incorrectly assumed that a router can forward announcements to itself. More broadly, by correctly modeling the low-level details of RFC 4271, our verified version of Bagpipe found 19 apparent errors in three ASes with over 240,000 lines of Cisco and Juniper BGP configuration. Bagpipe was also able to *guarantee* 4 properties of real ASes, and verify several textbook examples from Juniper's technical documentation. We formally verified Bagpipe's soundness in Coq, our proof totals 2960 lines of Coq code.

5. C-BGP

We evaluated our BGP semantics by randomized differential testing [13] against C-BGP [33], a popular open-source BGP simulator. In randomized differential testing, randomly generated test cases are run in multiple implementations of a system and their output is compared; differences indicate possible bugs. Our application of randomized differential testing both increases confidence in the semantics' correctness and demonstrates the semantics' broader utility by identifying a bug in C-BGP.

C-BGP enables network engineers to test configurations without running them on real hardware. It simulates a group of routers from any number of ASes running the BGP protocol, and allows users to observe how announcements are exchanged in the network. Users configure C-BGP by describing a network's topology, and by inputting router configurations in a format similar to the one accepted by Cisco routers. C-BGP is used by network administrators to determine the effects of changes to their simulation or topology; it is therefore important that it faithfully reflects the BGP specification.

To test C-BGP (and potentially other BGP simulators), we developed a program in Coq, based on our semantics, which

checks sequences of BGP events (deliveries of update or withdraw messages) to ensure that they are permitted by our semantics of the BGP specification. We then used Coq's extraction facility to extract this program to Haskell. We wrote a parser for C-BGP's trace format in order to convert C-BGP's text traces to checkable sequences of events. We then used the QuickCheck random testing tool [9] to write generators for random network topologies and BGP configurations, which we convert to both C-BGP's input format and to the corresponding functions in our BGP semantics (e.g., *imp*). This enables us to check C-BGP traces on randomly-generated inputs to determine whether they agree with our semantics.

We ran this differential testing tool over 100,000 times, on randomly generated topologies of moderate size and configurations of moderate complexity. We did not discover any bugs in our semantics, though we did identify bugs in the differential tester itself. Using test cases where C-BGP disagreed with our semantics, we discovered two issues in C-BGP.

When a router's export filters take two messages m_{i1} and m_{i2} to the same message m_o, the router will send m_o twice in a row to the same neighbor if it receives both m_{i1} and m_{i2}, in violation of Section 9.2 of the BGP specification RFC 4271 [34]. We reported this bug to the maintainer, who acknowledged it as a bug. We have submitted a fix to the C-BGP maintainers.

In the course of debugging our differential tester, we discovered another issue with C-BGP: rejected update messages are not placed in a router's *adjRIBsIn*, leading to a confusing case in which a router receives a withdraw message when there is no corresponding update message in its *adjRIBsIn*. A comment in the C-BGP source code where such a withdraw is handled indicates that the developers thought that any case where such a message is delivered is a bug. While this bug does not lead to incorrect observable behavior (since it does not affect the trace of delivered messages), it is a violation of the BGP specification.

The fact that C-BGP and our semantics agree on most test cases provides evidence that our semantics correctly reflects the real world.

6. Related Work

In this section, we address related work on BGP formalisms, checkers, simulators, and software-defined networking.

BGP Formalisms The Stable Paths Problem (SPP) [20] is a simplified model of BGP for which many theoretical results have been proven, including that solving SPPs is PSPACE hard [8]. In contrast to our semantics, SPP does not model all required features of RFC 4271. For example, SPP does not model routers within an AS, multiple connections between ASes, Routing Information Bases, update messages and all their attributes, the full route selection algorithm, update message withdrawals, and multiple ASes injecting a route for the same prefix; all these features are frequently used in practice. Extensions exist to SPP [21, 19, 37] that mitigate some (but not all) of these limitations.

Gao & Rexford [17]'s proof of Internet-wide route convergence is based on a simplified model of BGP which, in contrast to our semantics, does not model all required features of RFC 4271. Gao & Rexford's proof has also been adapted to SPP [16].

Andreas Voellmy used Isabelle/HOL to formalize a simplified model of BGP's operation at the AS level [40]. Similar to SPP, it does not model the behavior of individual routers or communication within an AS. This model was used to verify one policy for one textbook example configuration.

The Formally Verifiable Routing (FVR) project [42, 41, 43] provides a formal algebra for reasoning about BGP properties (e.g. convergence). FVR's formalism is based on the SPP semantics and thus has the same limitations.

BGP Checkers and Compilers Propane [4] provides a high-level language that BGP administrators can use to specify how packets should be routed though the network. Propane then compiles specifications written in this language to a collection of BGP router configurations. This compilation step is currently not verified. We believe that our semantics could be used to verify the compilation step of Propane, and similar tools like Nettle [39], to guarantee that their generated BGP configurations are correct by construction.

Batfish [15] is a Datalog-based BGP configuration checker, that translates router configurations and a topology description into Datalog facts. Given these facts, Batfish employs a set of Datalog rules to populate each router's routing tables. Batfish's model of the BGP data-plane is quite accurate, and can be used to test properties (e.g. network convergence) given a particular set of router configurations and received BGP announcements. However, unlike our semantics, it cannot be used to verify properties for all configurations and all announcements in the control plane.

rcc [14] is a BGP configuration checker; notable for its adoption by AS administrators, and finding a large number of router misconfigurations. rcc infers inter-AS relationships from routers configurations to find violations of route validity and path visibility. The tool is not based on a formal model of BGP, and reports both false positives and false negatives.

BGP networks are often symmetrical, which allows BGP checkers to exploit techniques recently developed by Plotkin et al. [32]. Our semantics could be used to gain confidence in the correctness of such checkers.

BGP Simulators There exist many BGP simulators [33, 35, 31, 28], that given a topology and a set of configurations, determine how traffic will be routed. Network administrators can use simulators both for debugging existing problems and for testing potential new configurations. Our semantics can be used to test simulators, as well as actual implementations of BGP routers, which have been bug prone in the past [26].

SDN Software defined networking (SDN) is a new paradigm for intra-domain routing (routing within an AS). With SDN, routing is controlled by a single program running on a master router, instead of the interplay between a multitude of individually configured routers. Much work has been devoted to verifying the behavior of SDNs, especially on the data plane, including language support [30], model-checking [2, 12], and full formal verification [22, 1]. By providing a semantics for BGP, we hope to enable similar achievements for inter-domain routing (routing between ASes) and the control plane.

7. Conclusion

This paper presented the first mechanized formal semantics of the BGP specification RFC 4271. The semantics is implemented in Coq. Our semantics models all required features of the BGP specification modulo low-level details such as bit representation of update messages and TCP.

Three case studies showed how to use our semantics to develop reliable proofs, checkers, and simulators (these case studies also provided evidence for the correctness of our semantics). 1) We formalized and extended the seminal pen-and-paper proof by Gao & Rexford on the convergence of BGP, revealing necessary extensions to Gao & Rexford's original assumptions. 2) We verified the soundness of the Bagpipe tool which automatically checks that BGP configurations adhere to given specifications. 3) We tested the popular BGP simulator C-BGP against our semantics, revealing one bug in C-BGP.

References

[1] C. J. Anderson et al. "NetKAT: Semantic Foundations for Networks". In: *POPL*. 2014.

[2] T. Ball et al. “VeriCon: Towards Verifying Controller Programs in Software-defined Networks”. In: *PLDI*. 2014.

[3] T. Bates, E. Chen, and R. Chandra. *BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)*. RFC 4456. 2006.

[4] R. Beckett et al. “Don’t Mind the Gap: Bridging Network-wide Objectives and Device-level Configurations”. In: *SIGCOMM*. 2016.

[5] *BGP Feature Guide for the OCX Series*. 2015.

[6] M. Brown. *Pakistan hijacks YouTube*. http://research.dyn.com/2008/02/pakistan-hijacks-youtube-1/. 2008.

[7] R. Chandra, P. Traina, and T. Li. *BGP Communities Attribute*. RFC 1997. 1996.

[8] M. Chiesa et al. “Using routers to build logic circuits: How powerful is BGP?” In: *ICNP*. 2013.

[9] K. Claessen and J. Hughes. “QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs”. In: *ICFP*. 2000.

[10] R. Coltun et al. *OSPF for IPv6*. RFC 5340. 2008.

[11] J. Cowie. *China’s 18-Minute Mystery*. http://research.dyn.com/2010/11/chinas-18-minute-mystery/. 2010.

[12] M. Dobrescu and K. Argyraki. “Software Dataplane Verification”. In: *NSDI*. 2014.

[13] R. B. Evans and A. Savoia. “Differential Testing: A New Approach to Change Detection”. In: *ESEC-FSE*. 2007.

[14] N. Feamster and H. Balakrishnan. “Detecting BGP Configuration Faults with Static Analysis”. In: *NSDI*. 2005.

[15] A. Fogel et al. “A General Approach to Network Configuration Analysis”. In: *NSDI*. 2015.

[16] L. Gao, T. G. Griffin, and J. Rexford. “Inherently safe backup routing with BGP”. In: *INFOCOM*. 2001.

[17] L. Gao and J. Rexford. “Stable Internet Routing Without Global Coordination”. In: *SIGMETRICS*. 2000.

[18] S. Goldberg. “Why Is It Taking So Long to Secure Internet Routing?” In: *Queue* (2014).

[19] T. G. Griffin, F. B. Shepherd, and G. Wilfong. “Policy disputes in path-vector protocols”. In: *ICNP*. 1999.

[20] T. G. Griffin, F. B. Shepherd, and G. Wilfong. “The Stable Paths Problem and Interdomain Routing”. In: *TON* (2002).

[21] T. G. Griffin and J. L. Sobrinho. “Metarouting”. In: *SIGCOMM*. 2005.

[22] A. Guha, M. Reitblatt, and N. Foster. “Machine-verified Network Controllers”. In: *PLDI*. 2013.

[23] *Internet2 Configurations*. http://vn.grnoc.iu.edu/Internet2/configs/configs.html.

[24] *Junos OS: Routing Policies, Firewall Filters, and Traffic Policers Feature Guide for Routing Devices*. 2016.

[25] D. Madory. *Chinese Routing Errors Redirect Russian Traffic*. http://research.dyn.com/2014/11/chinese-routing-errors-redirect-russian-traffic/. 2014.

[26] P. Mah. *BGP bug found in Juniper router software*. http://www.techrepublic.com/blog/it-news-digest/bgp-bug-found-in-juniper-router-software/. 2007.

[27] D. McConnell. *Chinese company ‘hijacked’ U.S. web traffic*. http://www.cnn.com/2010/US/11/17/websites.chinese.servers/. 2010.

[28] P. McDaniel and K. Butler. “Testing Large Scale BGP Security in Replayable Network Environments”. In: *DETER*. 2006.

[29] D. Meyer, J. Schmitz, and C. Alaettinoglu. *Application of Routing Policy Specification Language (RPSL) on the Internet*. 1997.

[30] C. Monsanto et al. “A Compiler and Run-time System for Network Programming Languages”. In: *POPL*. 2012.

[31] *NetSim: Network Simulator*. http://www.boson.com/netsim-cisco-network-simulator.

[32] G. D. Plotkin et al. “Scaling Network Verification Using Symmetry and Surgery”. In: *POPL*. 2016.

[33] B. Quoitin and S. Uhlig. “Modeling the Routing of an Autonomous System with C-BGP”. In: *IEEE Network* (2005).

[34] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. 2006.

[35] G. F. Riley. “Large-scale network simulations with GTNetS”. In: *WSC*. 2003.

[36] D. Slane. *2010 Report to Congress of the U.S.–China Economic and Security Review Commission*. 2010.

[37] M. Suchara, A. Fabrikant, and J. Rexford. “BGP safety with spurious updates”. In: *INFOCOM*. 2011.

[38] D. Turner et al. “California Fault Lines: Understanding the Causes and Impact of Network Failures”. In: *SIGCOMM*. 2010.

[39] A. Voellmy and P. Hudak. “Nettle: A Language for Configuring Routing Networks”. In: *DSL*. 2009.

[40] A. R. Voellmy. “Proof of an interdomain policy: a load-balancing multi-homed network”. In: *SafeConfig*. 2009.

[41] A. Wang et al. “Analyzing BGP Instances in Maude”. In: *FORTE*. 2011.

[42] A. Wang et al. “Formally Verifiable Networking”. In: *HotNets*. 2009.

[43] A. Wang et al. “FSR: Formal Analysis and Implementation Toolkit for Safe Inter-domain Routing”. In: *SIGCOMM*. 2011.

[44] K. Weitz et al. “Scalable Verification of Border Gateway Protocol Configurations with an SMT Solver”. In: *OOPSLA*. 2016.