Practical Verification Techniques for Wide-Area Routing

Nick Feamster
MIT Computer Science and Artificial Intelligence Laboratory
200 Technology Square, Cambridge, MA 02139
feamster@lcs.mit.edu

Abstract

Protocol and system designers use verification techniques to analyze a system's correctness properties. Network operators need verification techniques to ensure the "correct" operation of BGP. BGP's distributed dependencies cause small configuration mistakes or oversights to spur complex errors, which sometimes have devastating effects on global connectivity. These errors are often difficult to debug because they are sometimes only exposed by a specific message arrival pattern or failure scenario.

This paper presents an approach to BGP verification that is primarily based on static analysis of router configuration. We argue that: (1) because BGP's configuration affects its fundamental behavior, verification is a *program analysis* problem, (2) BGP's complex, dynamic interactions are difficult to abstract and impossible to enumerate, which precludes existing verification techniques, (3) because of BGP's flexible, policy-based configuration, some aspects of BGP configuration must be checked against a higher-level specification of *intended* policy, and (4) although static analysis can catch many configuration errors, simulation and emulation are also necessary to determine the precise scenarios that could expose errors at runtime. Based on these observations, we propose the design of a BGP verification tool, discuss how it could be applied in practice, and describe future research challenges.

1. Overview and Motivation

Networks establish global reachability by exchanging information using the Border Gateway Protocol v4 (BGP) [18]. Operators need BGP's configuration flexibility to achieve tasks in a wide variety of network settings, but this flexibility also implies that misconfiguration can cause the protocol to operate incorrectly. Correct routing depends on the consistent configuration of routers both within an AS and across ASes; thus, minor localized errors can cause BGP's behavior to violate properties that are fundamental to globally correct routing.

Most large ISPs contain several hundred routers, and the latest Cisco IOS release contains over 7,000 configuration commands, many of which can be configured with arbitrary parameters. Given this complexity, mistakes are frequent and expected. While existing tools facilitate some degree of configuration management [3, 9], operators must frequently perform configuration tasks manually in low-level, vendor-specific configuration languages [19].

As a result, misconfiguration is an all-too-frequent occurrence that often disrupts connectivity [15]. Even minor aspects of local configuration can influence global Internet connectivity. For example, in 1997, a small ISP blackholed a significant portion of Internet destinations when a misconfigured router announced a large portion of prefixes in the Internet's routing table as destinations inside its own network [4]. This incident, now notorious as "AS 7007", resulted from a configuration that "redistributed" routes from that AS's interior routing protocol into BGP; as a result, much of the Internet became unreachable. One week in July 2003 alone saw several cases of local misconfiguration that caused global events, including: (1) a misconfigured router in a small ISP caused it to become a transit network for two tier-1 ISPs [17], and (2) a small European ISP was assigning IP addresses to routers from unassigned address space [2]. The ability to verify BGP configuration before deployment most likely would have prevented these mishaps.

Today, network operators, protocol designers, and researchers typically reason about BGP's behavior by observing the effects of a particular configuration on a live network. The state of the art for router configuration typically involves logging configuration changes and rolling back to a previous version when a problem arises [3, 9]. This approach is not effective. First, the symptoms of a configuration error may appear long after the configuration error was introduced. For example, a network operator may not notice a misconfigured route filter until a route advertisement that should have been filtered is advertised. Therefore, operators may find reverting the configuration difficult, since the appearance of an error is often not correlated with the configuration change that caused it. Second, although reverting to a previous configuration may solve the problem, it does not explain *why* the configuration was wrong in the first place; this means that an operator may eventually reintroduce the error, especially if the error is conceptual.

The lack of a formal reasoning framework means that router configuration is time-consuming and ad hoc. Furthermore, today's routing configuration is based on the manipulation of low-level mechanisms (e.g., access control lists, import and export filters, etc.), which makes routing configuration tedious, error-prone, and difficult to reason about. Network operators need tools based on systematic *verification techniques* to ensure that BGP's operational behavior is consistent with the intended behavior (i.e., that the network is operating "correctly"). We propose a verification tool that helps operators and protocol designers reason about high-level properties of routing protocols.

We believe that wide-area routing will always need verification tools, even as configuration languages improve. Although some configuration errors are caused by simple typographical errors, many configuration errors are *conceptual*. The policy-based nature of wide-area routing configuration allows operators to accommodate a wide variety of conditions and network engineering tasks, but this flexibility also enables operators to make catastrophic mistakes. As long as this possibility exists, operators will need mechanisms to check the correctness of routing configuration.

Verification implies a notion of *correctness*; for traditional programs and protocols, correctness simply requires adherence to a specification. Our *routing logic*—a set of rules that facilitates reasoning about whether a routing protocol satisfies a particular property or set of properties—suggests one possible notion of correctness for wide-area routing [6]. The logic classifies "correctness" in terms of five properties: *validity* (the existence of a route implies the existence of a corresponding path), *visibility* (the existence of a path implies the existence of a corresponding route), *safety* (the existence of a stable, unique path assignment), *determinism* (best route selection is independent of message ordering and the presence of sub-optimal routes), and *information-flow control* (the protocol conforms to a specified information flow policy; that is, it does not "leak" information). Although many aspects of BGP configuration are clearly correct or incorrect, correctness is often determined by how well the actual configuration conforms to the *intended behavior*. For example, an operator might specify that routes heard from one AS should not be readvertised to another; a verifier should check that access control lists, import and export policies, and communities across multiple routers correctly implement this policy.

BGP's flexible configuration makes it much more like a distributed program than a traditional protocol. BGP's configuration determines which (and whether) routing messages are distributed; therefore, BGP verification *must* analyze the configuration.

We believe that a combination of static analysis and offline simulation or emulation, which we call a *sandbox*, is the right approach to BGP verification. Static analysis is useful for verifying systems that are inherently difficult to model, which makes it appealing for verifying BGP. Applying static analysis to BGP verification involves examining the static configuration of all routers within an AS and applying a set of rules to check for inconsistencies. This approach poses several challenges. We must identify the configuration aspects that can potentially affect correctness and design methods for expressing and testing correctness rules. In cases where the notion of "correctness" is not hard and fast, static analysis should at least verify that BGP's configuration conforms to the intended behavior. These challenges are exacerbated by the fact that BGP configuration is distributed across many routers. Furthermore, because BGP's behavior depends on the route advertisements it receives as well as failure scenarios, static analysis alone often cannot determine whether a BGP configuration will cause incorrect behavior. We believe that a sandbox can use inputs suggested by static analysis to help operators identify problematic scenarios and ask "what-if" questions about BGP configuration.

In this paper, we propose the design of a BGP verifier that a network operator could use to test possible configurations offline, prior to deploying them on a live network. We primarily focus on unintentional configuration errors that result from configuration inconsistencies *within a single AS* because, in practice, network operators typically only have access to their own configuration. Verification could use external information (e.g., from a database [10] or policy registry [11]) to find data inconsistencies or inter-AS policy conflicts, but we focus on relatively language-independent configuration aspects that can be verified using configuration from one AS with no additional information. Additionally, we discuss how aspects of our work may provide insights that help improve future configuration languages.

This paper presents the case for new, domain-specific BGP verification techniques (Section 2), identifies and classifies how these techniques can check configuration aspects that affect BGP's correctness (Section 3), and proposes an approach to verifying BGP configuration using static analysis and sandboxing (Section 4).

2. Practical BGP Verification Techniques

In this section, we explain why we pursue static analysis, rather than model checking, to identify errors. Since BGP's behavior depends on dynamic conditions (e.g., link failures, advertisements from neighboring ASes, etc.), static analysis alone will not catch all configuration errors. We argue that static analysis should be combined with a *BGP sandbox*—simulation or emulation that describes BGP's behavior under various dynamic conditions.

2.1 The case for static analysis

A BGP verifier should efficiently identify cases where BGP violates some correctness property. Although BGP operates much like a distributed program, most program verification techniques are not directly applicable. Theorem proving facilitates reasoning about high-level system properties but is inefficient and often manual; verification must be automatic to provide timely feedback to network operators. Model checking has been popular for examining the correctness of traditional communications protocols with well-defined state machines (e.g., TCP), so it might seem like a reasonable approach. Unfortunately, applying model checking to wide-area routing presents two serious difficulties. First, it requires state space exhaustion, which is tedious—all possible message orderings, prefix combinations, AS paths, and so on—and BGP's complexity makes it difficult to collapse these states. More importantly, *the policy-based nature of wide-area routing makes it impossible to ever know every possible state*.

Figure 1 helps explain why model checking BGP is hard to get right. In this example, AS 0 wants to load balance traffic between its two links to AS 1. The router s sends traffic via the default exit route unless it hears a route advertisement from the shaded router. The shaded router only advertises routes to s that match the AS path regular expression "1 [0-9]+ 3"; furthermore, it is misconfigured and readvertising invalid routes to s. Let's assume that it is incorrectly setting the next-hop when readvertising routes. In this situation, the shaded router only advertises invalid routes if AS 1 switches over to a backup AS. Since AS path "1 2 3" is a backup path for AS 1, it would not be visible to AS 0 under normal circumstances. Thus, model checking AS 0's configuration *may fail to explore the very states that expose bugs*. Model checking might be able to able to verify some aspects of BGP correctness under certain simplifying assumptions, but subtleties make model checking BGP almost as complex as BGP itself.

Static analysis, which tests configurations against constraints that correspond closely to the configuration itself, is a simpler approach to BGP verification. By analyzing only the configurations, static analysis could identify that the shaded router was readvertising routes with unreachable next-hop values. In Section 3.1, we discuss the configuration details that cause the incorrect behavior in Figure 1 and describe how static analysis can catch this error.

2.2 The case for a BGP sandbox

Static analysis can efficiently identify configurations that may cause BGP's correctness to be violated, but BGP's behavior also depends on dynamics, which may expose emergent incorrect behavior from a seemingly correct configuration. For example, a network operator may want to verify that the network operates "correctly" during link failures (or more serious disasters), fluctuations in traffic volumes, planned maintenance, or instances where a neighboring AS suddenly begins advertising new routes. A *BGP sandbox* (i.e., a simulation or emulation tool that models BGP's behavior, given its configuration [8]) explores these interactions and can help operators perform higher-level tasks (e.g., traffic engineering) correctly.

Obviously, verifying BGP configuration with a sandbox alone is not tractable because the space of possible inputs is too large. Static analysis can *prune the input space* to include only inputs that are most likely to expose incorrect behavior. Additionally, to simplify

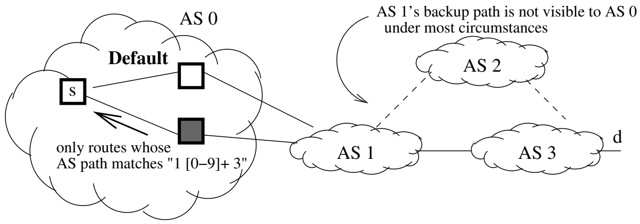

Figure 1: AS 0 cannot use model checking to verify configuration because some states, like backup paths, may be hidden.

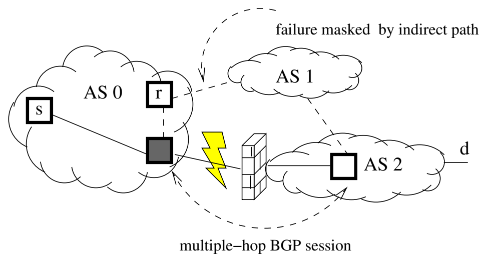

Figure 2: A sandbox can expose emergent incorrect behavior.

the process of modeling the protocol, a sandbox will typically require that certain configuration constraints are satisfied; static analysis can check these types of constraints.

Figure 2 helps explain why operators need a BGP sandbox to expose certain correctness violations. In this example, the shaded router is configured to establish a BGP session over multiple network hops (this can be useful if AS 2's router is behind a firewall). The shaded router advertises a route to d via AS 2. If the direct path between AS 0 and AS 2 fails, the BGP session between the shaded router and the firewalled router may remain active if an alternate multiple-hop path between the two routers exists. Thus, the shaded router perceives that its path to AS 2 is active, and continues to advertise a route to d directly via AS 2. Static analysis could discover that a particular router configuration might raise this possibility (thus pruning the input space), but a BGP sandbox can help determine whether (and when) it could actually occur by modeling BGP's behavior under specific failure scenarios.

3. Verifying BGP Configuration

To verify BGP's correctness, we must first understand how various configuration aspects affect each correctness property. We use the routing logic properties from our previous work to classify how various configuration statements affect aspects of correctness [6]. Each subsection focuses on one or two correctness properties and presents one or two examples of configuration statements that affect those properties. Table 1 summarizes the effects of many more configuration statements on higher-level properties; a BGP verifier should express constraints related to each of these statements that guarantee that the higher level property holds. For each example that we discuss, we present a potential misconfiguration and specify the constraint that should be satisfied to avoid the error. We classify the constraints in terms of the following questions:

- Does verifying the constraint involve checking configuration at a single router or multiple routers?
- Does it require information from other routing protocols (e.g., from the IGP)?
- Does it require information that is external to the configuration (e.g., from a configuration database)?
- Does it require configurations from multiple ASes?
- Can it be specified as a necessary condition, a sufficient condition, or both?
- Does it require the operator to specify the *intended* correct behavior, or can it be verified without this specification?
- Can static analysis alone can check the constraint, or is a BGP sandbox is also necessary?

The examples we present are motivated by options in Cisco's IOS configuration language, but other languages have similar options, and the examples we discuss are language-independent. Although the examples are not exhaustive, other studies [15] and personal communication with operators have helped us focus on representative errors that are common, elusive, or catastrophic.

3.1 Validity and Visibility

Validity requires that a route for a certain destination correspond to the actual path that packets sent along that route would follow; it is important for correctness because an invalid route to a destination can prevent packets from reaching their intended destination (e.g., by creating a blackhole, forwarding loop, etc.). Visibility means that a valid path will have a corresponding route.

Each BGP route contains a next-hop attribute that tells a router the IP address to forward packets to in order to use that route; configuration affects how this attribute is set. eBGP-speaking routers typically enable a configuration option that sets the BGP next-hop attribute to the loopback address of that eBGP-speaking router (rather than the address of the router in another AS) to ensure that the next-hop addresses advertised by iBGP are in the internal routing protocol ("IGP") and, thus, that internal routers can use the advertised route (see `next-hop-self` in Table 1). This option simplifies configuration but is not always appropriate [1].

The following necessary condition verifies that eBGP-speaking routers will advertise routes with reachable next-hop IP addresses: if an eBGP-speaking router does *not* set the next-hop attribute to its own loopback address, then valid routing requires that the next-hop from the neighboring AS's end of the eBGP session be incorporated into the IGP; in this case, routes for these addresses *must not be readvertised to other eBGP peers*. This constraint ensures that internal routers have a route to the next-hop via the IGP and that the internal routers do not advertise these routes to neighboring ASes. The error shown in Figure 1 could arise precisely because the shaded router did not reset the BGP next-hop attribute, and the IP addresses from its session to AS 1 were not in the IGP. According to several network operators, this error frequently appears in practice. Static analysis can verify this necessary condition using configurations across multiple routers within a single AS and some IGP configuration information; we discuss the specifics of how static analysis can check this constraint in Section 4.

Network operators typically use a logical full-mesh of iBGP sessions between routers to ensure that every router in the AS receives the same set of BGP routes, but route reflection is often used to achieve intra-AS consistency more scalably. This technique is subtle, involves many configuration commands, and can violate validity and visibility if misconfigured. The best common practice for correct route reflector configuration requires satisfying sufficient conditions that involve both iBGP and *IGP* configuration [6, 13]. Static analysis can use BGP and IGP configuration from the routers in the local AS to verify that these conditions are satisfied.

Visibility can be violated if the iBGP signaling graph becomes partitioned; this can occur for several reasons. For example, the AS may not have "full mesh" iBGP, or route reflector clusters may be misconfigured. While these types of fundamental partitions can

Statement or Clause	*What it does*	*How it affects correctness*
	VALIDITY AND VISIBILITY	
neighbor <IP> [options]	attempts to establish BGP session to IP with specified options	1. IP address or options mismatch 2. iBGP session failures to a router interface rather than loopback address are not masked by IGP
no synchronization	allows iBGP routes to appear in BGP tables without appearing in IGP (external routes need not be in IGP)	a non-iBGP-speaking router between two iBGP routers creates a black-hole
ebgp-multihop	allows a router to establish eBGP with a router to which it is not directly connected (useful for routers behind firewalls, border routers that can't handle full routing tables, etc.)	inconsistency results if the path between the routers fails over to a more circuitous route (perhaps through another AS)
next-hop-self	sets the next-hop IP address for readvertised routes to the IP address of the current router; often used when eBGP-speaking routers readvertise externally-learned routes via iBGP; sometimes disabled for reliability reasons	if *not* used, next-hop IP addresses for externally-learned routes must be injected into the IGP (and also *not* readvertised to other ASes)
set path prepend <AS>	prepend additional AS hops to the AS path of readvertised routes	prepending with arbitrary AS numbers results in invalid routes
network <prefix>	originates ("injects") routes for prefixes	potential for bogus route injection
community-list, access-list	determines the prefixes for which advertisements are accepted and readvertised; facilitates route filtering	improper configuration can accept or leak bogus (or unwanted) routes
redistribute [bgp|ospf|connected|static]	imports routes from one routing protocol into another; useful for various reasons (e.g., if some routers don't support BGP, etc.)	blackholing is likely and commonplace
aggregate-address [addr]-summary-only	aggregates contiguous prefixes in a certain range; reduces the number of advertised prefixes, thereby reducing routing table size and improving scalability	potentially causes fate sharing between independent networks; impedes traffic engineering
	SAFETY AND DETERMINISM	
always-compare-med	causes the MED attribute to be compared across all routes, rather than only between routes from the same AS; not commonly used	if not used, then various constraints must be satisfied to prevent route oscillation
deterministic-med	causes the most recently received route to be compared with the set of all available routes for a prefix, rather than just the current best route; sometimes (but rarely) disabled to save CPU/state	if not used, route selection depends on route advertisement arrival order
best path compare-routerid	causes route selection to skip criterion that prefers the route that was received first; sometimes not used to prefer "stable" routes	if not used (and maximum-paths does not enable multiple best routes), route selection depends on route advertisement arrival order
	INFORMATION-FLOW CONTROL	
set community, access-list, match community-list	controls route propagation	incorrectly specified or unspecified ACLs, import policies, or export policies can cause routes to leak

Table 1: Some examples of BGP configuration statements that affect high-level correctness properties.

be detected with static analysis of local BGP configuration alone, many partitions may only appear if a BGP session fails or certain routes are filtered or suppressed. In these types of situations, a sandbox could highlight the precise failure scenarios that cause a partition (hence, a visibility violation) to occur.

Other constraints related to validity and visibility require more extensive information, such as configurations from neighboring ASes or knowledge about operator intent. Static analysis requires configuration from neighboring ISPs to determine that one of these ISPs has misconfigured a filter and is thus preventing some downstream network from being visible to the rest of the Internet. Other configuration options may require a more detailed specification of operator intent. For example, aggregating contiguous prefix blocks is a common practice to reduce routing table size, but an operator might *intentionally* advertise separate contiguous prefixes for traffic engineering purposes. In this case, static analysis must compare the configuration with a specification of the operator's intent to determine whether the configuration is correct.

3.2 Safety and Determinism

Safety requires that, given a set of routing choices, BGP should eventually arrive at a stable path assignment; in other words, there should be no oscillations induced by the configuration itself. Most safety issues involve inter-AS policy, but improper route reflector configuration can also cause intra-AS route oscillation induced by BGP's "MED" attribute [12, 16]. Any one of several sufficient conditions can prevent this behavior: (1) inter-cluster IGP metrics are higher than intra-cluster IGP metrics, (2) MEDs are disabled completely, (3) MEDs are compared across all routes (as opposed to routes from the same AS only), (4) the AS uses full-mesh iBGP. Even if none of these conditions are satisfied, MED oscillation may not occur—the anomaly depends on a specific sequence of route advertisements. Static analysis of BGP and IGP configuration from a single AS can determine whether these sufficient conditions are satisfied, and, if the conditions are not satisfied, a sandbox can determine the sequence of advertisements that will expose the anomaly. However, verifying that safety is satisfied *globally* (i.e., that policy-based oscillations between multiple ASes do not occur) requires static analysis of configurations from multiple ASes to ensure that certain sufficient conditions are satisfied [14].

Determinism requires that the routing protocol behave the same way under (1) different orderings of route messages (*time immunity*) and (2) different sets of available routes (*set immunity*) [6]. Unless an operator has intentionally configured nondeterminism, BGP should satisfy determinism, because it facilitates debugging, as well as the use of traffic engineering tools [7]. Table 1 lists several configuration options that should be enabled to ensure that BGP satisfies time immunity. Static analysis can verify these constraints by checking configurations at individual routers in the AS, as these options do not have dependencies across routers. Unfortunately, as long as operators enable the standard use of MED (where the MED value is compared only between routes from a single AS, rather than across all ASes), BGP will violate set immunity [6]. If the operator *intends* to disable MED, static analysis can easily verify that MED is disabled on all eBGP-speaking routers.

3.3 Information-flow Control

The information-flow control property requires the exchange of routing messages to conform to some higher-level information flow policy. An AS must control which routes propagate to neighboring ASes because advertising a route to another AS implies a willingness to carry traffic for that AS. In practice, an AS can control unintended route propagation using import and export policies in conjunction with the "community" attribute. Routers often assign a community to routes received from other ASes based on which session the route was learned from (e.g., from a peer, customer,

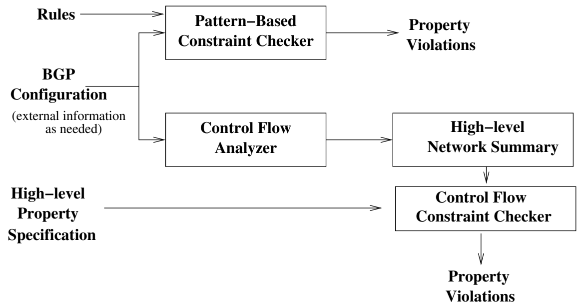

Figure 3: A static constraint checker for BGP configuration.

etc.). These routes are readvertised to other routers within the AS along with the assigned community attribute, and other routers in the AS determine whether to readvertise the route to a particular neighbor based on this attribute. Effectively, network operators use low-level mechanisms like communities to specify a high-level policy (e.g., don't advertise routes from one peer to another); this approach makes misconfiguration much more likely. Static analysis of BGP configuration, along with a specification of operator intent, can ensure that the configuration satisfies high-level information flow policy. In general, static analysis can verify these types of information-flow policies using BGP configurations from all of the routers inside one AS. We describe more details of how an actual static analysis tool might verify such policies in Section 4.

4. Applying Verification Techniques to BGP

In this section, we describe a practical approach to BGP verification that combines static analysis and a BGP sandbox. Static analysis parses configuration statements to detect errors that are evident from the configuration commands themselves. A sandbox can determine whether (and under what circumstances) a seemingly correct configuration can produce incorrect behavior.

4.1 Verifying BGP with Static Analysis

We now discuss how to apply static analysis to BGP. We propose the design for a tool that uses static analysis to catch various types of BGP configuration errors that span multiple routers *within a single AS*. We first present a systematic approach for enumerating correctness constraints using properties of the routing logic. We then describe some practical methods for checking these constraints.

Configuration dependencies determine how static analysis should check constraints on BGP configuration. For example, validity tests verify that the BGP configuration does not readvertise invalid routes (outbound validity). This involves checking that outbound access control lists (ACLs) are configured appropriately (i.e., that the AS is not advertising bogus prefixes or origin ASes, etc.). Additionally, outbound validity depends on inbound validity—the verifier must ensure that routes learned via eBGP sessions are valid (inbound validity). This requires ACLs to be configured to reject invalid prefixes and routes and iBGP configuration to be consistent with the IGP topology. Several of these checks depend on BGP-level connectivity, which in turn requires verifying configuration of router interfaces and IGP-level connectivity.

Figure 3 provides an overview of a static constraint checker for BGP. We believe that the constraints that a BGP verifier should check to guarantee correctness fall into the following categories:

1. *Pattern-based constraints.* Rules that are expressed in terms of the configuration language itself. Pattern-based constraints are appropriate for expressing low-level constraints that involve specific configuration options.

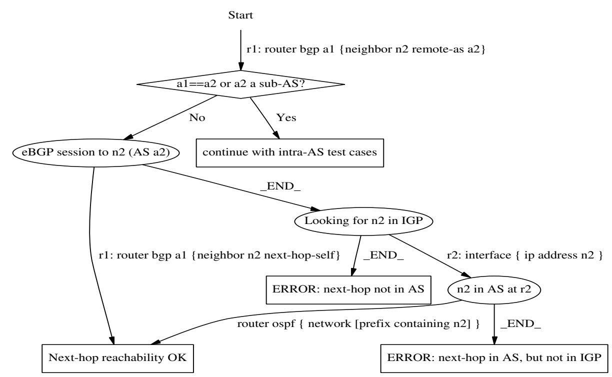

Figure 4: FSM for the next-hop reachability test. _END_ is taken when no other transition can be matched.

2. *Control-flow constraints.* Rules that express how routing messages should propagate with respect to routing protocols at lower scopes. For example, control-flow constraints can specify how iBGP routing messages should flow with respect to an AS's IGP graph.

   These rules can also express (1) what information (i.e., routes) should be imported into the AS, (2) what information should be exported to other ASes, and to which ASes it should be sent, and (3) whether and how that information should be processed.

Once the verifier parses the configuration, each type of constraint can be language-independent. Control flow constraints can be analyzed without regard to configuration statements. Although pattern-based constraints may be expressed in terms of the configuration language, these rules could also be expressed in a vendor-independent representation.

Pattern-based rules can be expressed as a finite state machine (FSM) that matches on patterns in router configuration. One complication arises because BGP configuration is distributed across multiple routers. For example, for two routers to establish a BGP session, each router must have a `neighbor` statement for an interface on the other's router (preferably the loopback). Scoping each configuration statement with a unique identifier representing the router where that statement appears facilitates the expression of rules that depend on configurations at multiple routers.

Let's revisit the example from Section 2.1 (Figure 1), where router s receives invalid routes from the shaded router because the routes contain incorrect next-hop values. As previously mentioned, inbound validity depends on iBGP and IGP consistency, which in turn depends on next-hop reachability. Figure 4 shows an FSM that checks the next-hop reachability constraints that we described in Section 3.1. Static analysis could determine that the shaded router from Figure 1 has the error "next-hop not in AS"—it is advertising routes with a next-hop that other routers cannot reach. As in other pattern-based checkers, the rules are expressed as templates: from the start state, any router r_1 and neighbor n_1 will match; once bound, a variable maintains the same binding.

Control-flow constraints, on the other hand, are high-level rules that express how routing information should propagate. For example, information-flow control involves controlling how routes propagate between BGP sessions. Outbound filters typically check whether a route has a particular community value that was set by another router. These filters use a *mechanism* to achieve a high-level policy, but the actual mechanism that is used to control the propagation of routes is not as important as the *intention*. In these types of cases, pattern-based rules are not appropriate; rather, a rule should specify constraints on how routes should be propagated. For example, a constraint might specify that routes from learned from one peer should not be advertised to another peer; as shown in Figure 3, checking this constraint not only requires a summary of control flow, but also a high-level specification of correct control flow.

For information-flow control, a network operator specifies the meaning of "correct" control flow, but other high-level control flow constraints are not intention-based. For example, checking that iBGP configuration satisfies sufficient conditions to avoid partitions in the iBGP signaling graph or loops induced by route reflectors involves checking how BGP control flow relates to the IGP.

4.2 A BGP Sandbox

As we saw in Section 2.2, configuration options can affect BGP's correctness when combined with dynamic effects. We developed an emulator that uses BGP routing tables and router configurations to emulate BGP's route selection process for each router in an AS [8]. This emulator efficiently predicts the outcome of the BGP decision process given a certain BGP configuration across all routers in an AS. This tool could be used to expose potential configuration pitfalls such as those in Figure 2 by emulating the BGP decision process under different failure scenarios.

Because of the extremely large input space, the primary challenge in using a sandbox is finding the inputs that expose property violations. We believe that static analysis can parse configuration to identify potential property violations and prune the input space to those that will likely cause property violations.

5. Discussion and Research Agenda

Analyzing BGP's configuration with the combination of static analysis and a BGP sandbox can help verify BGP's correctness because its correctness is largely defined by its configuration. We have incorporated some of these verification techniques into a preliminary tool and have made it available to network operators for feedback [5]. This work is a first step in an important, generally unexplored area and poses many interesting, unanswered questions.

How can verification exploit beliefs about correct configuration to automatically derive rules for static analysis? Since static analysis requires checking router configuration against many rules, exhaustively specifying every possible rule is manual and tedious. To alleviate this problem, verification could exploit statistical *beliefs* about what correct configuration looks like. For example, if an AS has several hundred routers, and all but a few are configured in a certain way, more likely than not the aberrations are mistakes.

What classes of errors can be discovered with various verification techniques? Clearly, there is an enormous space of BGP configuration errors that we need to understand better. What types of errors require information external to the configuration? What types of errors cannot be found with static analysis alone? What classes of errors involve configuration consistency across neighboring ASes, and what is the minimum amount of information that needs to be exchanged to check for these errors?

Finally, configuration checking techniques can help network operators find errors in configuration, but these techniques are only treating the symptoms of a more fundamental problem: *today's routing configuration languages are based on low-level mechanisms, rather than operator intent*. A better approach would allow an operator to specify configuration in terms of high-level policy without having to worry about the details of how the policy is actually implemented. Because checking today's routing configuration requires understanding how to specify operator intent, we believe that configuration checking is not only useful in and of itself, but also that figuring out how to specify operator intent will facilitate the design of a BGP configuration language that focuses on operator intent, rather than mechanism.

Acknowledgments

I am very grateful to Hari Balakrishnan and Jennifer Rexford, who both provided invaluable guidance for this work. I also thank Tom Barron, Randy Bush, Avi Freedman, Dave Meyer, Dave Ward, and Russ White, who offered useful insights into current practices. Thanks also to Senthil Ayyasamy, Frans Kaashoek, Richard Mortier, the NMS group at MIT, and the reviewers for thoughtful feedback.

6. References

[1] BICKNELL, L. Re: transit across the ixs. `http://www.merit.edu/mail.archives/nanog/1999-02/msg00192.html`, February 1999.

[2] BUSH, R. It's 1918 in Bologna. `http://www.merit.edu/mail.archives/nanog/msg11169.html`, July 2003.

[3] BUSH, R., ET AL. Watching your router configurations and detecting those exciting little changes. `http://www.nanog.org/mtg-0310/rancid.html`, October 2003. Panel at NANOG 29.

[4] FARROW, R. Routing instability on the Internet. *Network Magazine* (March 4, 2002). `http://www.networkmagazine.com/article/NMG20020304S0007/2`.

[5] FEAMSTER, N., AND BALAKRISHNAN, H. A systematic approach to BGP configuration checking. `http://www.nanog.org/mtg-0310/feamster.html`, October 2003. NANOG 29.

[6] FEAMSTER, N., AND BALAKRISHNAN, H. Towards a logic for wide-area Internet routing. In *ACM SIGCOMM Workshop on Future Directions in Network Architecture* (Karlsruhe, Germany, Aug. 2003).

[7] FEAMSTER, N., BORKENHAGEN, J., AND REXFORD, J. Techniques for interdomain traffic engineering. *Computer Communications Review 33*, 5 (October 2003).

[8] FEAMSTER, N., WINICK, J., AND REXFORD, J. A model of BGP routing for network engineering. *In submission*, Nov. 2003.

[9] Goldwire Formulator, 2003. `http://www.goldwiretech.com/products/formulator.cfm`.

[10] GOTTLIEB, J., GREENBERG, A., REXFORD, J., AND WANG, J. Automated Provisioning of BGP Customers. *IEEE Network* (2003).

[11] GOVINDAN, R., ALAETTINOGLU, C., VARADHAN, K., AND ESTRIN, D. Route servers for inter-domain routing. *Networks and ISDN Systems 30* (1998), 1157–1174.

[12] GRIFFIN, T., AND WILFONG, G. Analysis of the MED oscillation problem in BGP. In *Proc. ICNP* (Paris, France, November 2002).

[13] GRIFFIN, T., AND WILFONG, G. On the correctness of IBGP configuration. In *Proc. ACM SIGCOMM* (Pittsburgh, PA, August 2002).

[14] GRIFFIN, T. G., SHEPHERD, F. B., , AND WILFONG, G. The stable paths problem and interdomain routing. *IEEE Transactions on Networking 10*, 1 (2002), 232–243.

[15] MAHAJAN, R., WETHERALL, D., AND ANDERSON, T. Understanding BGP misconfiguration. In *Proc. ACM SIGCOMM* (Aug. 2002), pp. 3–17.

[16] MCPHERSON, D., GILL, V., WALTON, D., AND RETANA, A. *Border Gateway Protocol (BGP) Persistent Route Oscillation Condition*. Internet Engineering Task Force, August 2002. RFC 3345.

[17] PAYNE, J. Filtering Customer BGP Sessions. `http://www.merit.edu/mail.archives/nanog/msg11184.html`, July 2003.

[18] REKHTER, Y., AND LI, T. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, 1995. RFC 1771.

[19] REXFORD, J. State of the art in router configuration. `http://www.merit.edu/mail.archives/nanog/2002-01/msg00265.html`, January 2002.