# Rethinking Routing Configuration: Beyond Stimulus-Response Reasoning

*Nick Feamster*

## BGP Configuration Today

BGP-speaking routers make routing decisions and propagate routing messages based on local configuration in the hope that the resulting path assignments will provide stable, global connectivity. BGP routing involves local decisions made by routers based on flexible policies, which means that BGP's behavior is largely defined by its *configuration.* Controlling BGP's behavior by manipulating router configuration is a double-edged sword: its flexibility allows operators to use BGP to accomplish a wide range of tasks, but it also means that misconfigurations can, and do, cause significant problems. Even one misconfiguration can cause a complex sequence of errors that adversely affect global connectivity and are difficult to debug.

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. 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 based on a new reasoning framework that helps operators and protocol designers reason about high-level properties of routing protocols. Eventually, routing configuration should be based on *design patterns* that facilitate high-level tasks by more accurately specifying the operator's intent. Understanding what high-level properties should be checked in today's configuration will help us move in that direction.

## A New Reasoning Framework

We previously proposed that routing protocols be classified in terms of the following properties:

- *Validity.* The existence of a route to a destination implies that a packet sent along the corresponding path will reach its intended destination.
- *Visibility.* The existence of a valid path to a destination from some origin implies that the origin knows about a route for that path.
- *Safety.* Given a set of routes and policies, a route assignment exists such that no participant wants to change its route in response to other participants' routes (Griffin *et al.*'s Stable Paths Problem).
- *Determinism.* Given a set of possible routes and a set of policies, the routing protocol should always converge to the same predictable set of routes.
- *Information-Flow Control.* Routing messages should not expose more information than is necessary to achieve the above requirements, subject to some information flow policy.

For each property, we define rules that, if satisfied, imply that the property is satisfied. We call this set of properties, together with the rules to reason about them, the *routing logic.* The routing logic facilitates a more effective approach to verifying routing protocol configuration. Thinking about BGP's behavior in terms of the routing logic allows us to identify the ways that configuration affects each of these properties and to express the constraints that are sufficient to guarantee that these properties are satisfied.

## Routing Configuration in the Future: Systematic, Verifiable, Simple

Because BGP's correctness depends almost entirely on how it is configured, network engineers need a thorough, methodical process for verifying that a candidate router configuration will behave correctly, *before* deploying that configuration on a live network. We propose a systematic approach to configuration checking that is based on verifying conformance to the high-level properties of the routing logic. For each property, we determine the aspects of configuration that affect these high-level properties and define specific constraints that can be checked against router configuration using static analysis techniques.

We believe that these constraints fall into three general 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 commands.

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.

3. *Information-flow constraints.* Rules that 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.

Some constraints are so low-level that they are best expressed in terms of the configuration itself, similar to how constraints are specified in conventional static checkers for systems code. For example, a visibility constraint might specify that a BGP session between two routers should be between the routers' loopback addresses (rather than to specific interfaces). This is most easily checked with a pattern-based rule that checks that the routers' configurations have the appropriate `interface` and `neighbor` statements.

Control-flow constraints express how routes should propagate *within* an AS. For example, interactions between iBGP and IGP can cause persistent forwarding loops if route reflectors and their clients are not configured properly. A network operator would want to ensure that higher level properties are satisfied (e.g., that no iBGP link traverses an IGP path that includes a client of a different route reflector). A control flow graph could express this constraint, and a verification tool could verify the control flow created by the actual configuration against the intended control flow.

Information-flow control constraints are best specified in terms of intended and actual data flow. To verify that information flow conforms to a certain policy, operators can compose a high-level specification of how they intend information (i.e., routes) to propagate through their network. For example, an operator might specify that routes from one peer should not be propagated to other peers. The verifier then analyzes the low-level configuration to determine whether some aspect of the configuration violates the intent.

### Beyond Static Constraint Checking: Beliefs, Sandboxes, and Synthesis

While static constraint checking is useful for verifying many aspects of router configuration, it has several shortcomings. First, since static analysis requires checking router configurations against many rules, exhaustively specifying every possible rule is manual and tedious. To alleviate this problem, BGP verification should also 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. Applying beliefs to router configuration might catch a variety of errors, from simple mistakes, such as missing statements, to more subtle errors, such as misconfigured policies.

Second, certain configuration pitfalls may only be exposed as a result of a specific message pattern or failure scenario and are not evident from static analysis alone. For example, visibility can be violated if a neighboring AS does not send routing advertisements with consistent attributes on all BGP sessions with its neighbor (a neighboring AS might do this if it were trying to trick that AS into using "cold potato" routing). An AS can partially defend itself against these attacks with defensive configuration (e.g., by actively setting the same MED, origin type, and next-hop on incoming routes), but precisely determining vulnerability requires emulating exactly what happens when its neighbors send it a particular set of of routes. Similarly, certain property violations may only arise under specific failure scenarios; for this reason, network operators need a *sandbox* (i.e., a dynamic simulation and emulation environment) to test configuration robustness.

Finally, configuration checking can help network operators find errors and mistakes 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.* For example, to specify that routes heard from one AS should not be re-advertised to another, a network operator must correctly define access control lists, import and export policies, and communities across multiple routers. A better approach would allow an operator to specify this 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 it will help us discover the design patterns that will be most useful for high-level protocol configuration.