A Systematic Approach to BGP Configuration Checking

Nick Feamster and Hari Balakrishnan
M.I.T. Computer Science and Artificial Intelligence Laboratory
{feamster,hari}@lcs.mit.edu

http://nms.lcs.mit.edu/bgp/
BGP Configuration Determines Its Behavior

- Route injection, redistribution, aggregation
- Import and export route maps
- Access control lists, filtering
- AS Path prepending
- Communities
- Next-hop settings
- Route flap damping
- Timer settings

*BGP is a distributed program.*

*We need practical *verification* techniques.*
"What happens if I tweak this import policy?"
"Let’s just readjust this IGP weight..."
"New customer attachment point? Some cut-and-paste will fix that!"

Some time later, some "strange behavior" appears.
(OOPS! Revert.)

- Network operators have a terrible "programming environment".
  - Configuration is ad hoc and painful.
  - Wastes operator time.
  - Suboptimal performance, angry customers.
Higher Level Reasoning About Configuration

- **Verify** the behavior of a particular configuration.
  - Check "correctness properties".
  - Check that the configuration conforms to intended behavior.

  *More than a band-aid fix! Useful for any router configuration language.*

- **Specify** configuration based on intended behavior.
  - Configuring low-level mechanisms is error-prone.
  - Specifying high-level intended behavior makes sense.
Example: Information-flow Control

Simple rule: don’t advertise routes from one peer to other peers.

“Announce p”
Today: Specifying Policy with Mechanism

Bad: Import/export route maps, ACLs, communities, etc.

```
neighbor 10.0.0.1 route-map IMPORT-A in
neighbor 10.0.0.1 route-map EXPORT-A out
neighbor 192.168.0.1 route-map IMPORT-C in
neighbor 192.168.0.1 route-map EXPORT-C out
ip community-list 1 permit 0:1000
route-map IMPORT-C permit 10
  set community IMPORT 0:1000
!
route-map EXPORT-A permit 10
  match community 1
!```

"Announce p"
**Other Information-flow Control Examples**

**Goal:** Verify that route advertisements conform to intended information-flow policy.

- Partial peering

- Controlling prefix propagation
  - Bogons
  - "No Export" prefixes

- Conditional advertisements

- Signalling (e.g., with communities)
Higher Level Reasoning about "Correctness"

- **Validity**: Does it advertise invalid routes?
  - Bogus route injection, persistent forwarding loops, etc.

- **Visibility**: Does every valid path have a route?
  - Session resets, missing sessions, damped routes, etc.

- **Safety**: Will it converge to a unique, stable answer?
  - Policy-induced oscillation

- **Determinism**: Answer depend on orderings, etc.?
  - Irrelevant route alternatives can affect outcomes.

- **Information-flow control**: Expose information?
  - Accidental route leaks to neighbors, etc.
Verifying Configuration "Correctness"

• **Why?** Unlike most protocols, BGP’s correctness depends heavily on how it is configured.

• **How?** Systematically, according to properties:
  ▶ enumerate aspects of configuration that affect it
  ▶ test that those aspects conform to certain rules

• **Limitations?** Some aspects involve cooperation across ASes; not really possible today.

  *That’s OK, plenty goes wrong inside of one AS, too.*
We are developing a tool that checks correctness constraints for configuration.

RoLex (Routing Lexer)
RoLex: Configuration Verification Suite

- Two distinct parts that parse IOS configs.
  - Pattern-based constraint checker
  - Control flow analyzer

Send requests for more tools, features, etc!

http://nms.lcs.mit.edu/bgp/rolex/
Pattern-Based Rule Checker: Usage

- **Easy:** simple as running a script
  
  ```bash
  cd rolex/perl/src/pattern-rules/tests/
  ./nh-reachability-test.pl (or whatever)
  ```

- Chomps on all configs at once.

- Running time depends on network size, test, etc.
Pattern-Based Rule Checker: Sample Output

● Validity Test

  found ebgp on atlga-gw1 (AS 65000)
  ebgp: no next-hop-self atlga-gw1 (@10.215.0.113@)
  ERROR: @10.215.0.113@ not in iBGP/IGP (eBGP session)
  ...

● Visibility Test

  ERROR: no _r2 with loopback @10.0.1.65@ (from bosma-gw)
  ERROR: no _r2 with loopback @10.123.197.110@ (from bosma-rr1)
  ERROR: no _r2 with loopback @10.123.197.110@ (from bosma-rr2)
  ERROR: laxca-gw has NO "router bgp" statement
  ...

● Determinism Test

  atlga-rr2: deterministic-med OK
  ERROR: atlga-gw2 has no deterministic-med
  ERROR: attga-gw3 has no deterministic-med
  ...
  wswdc-rr1: compare-routerid OK
  ERROR: wswdc-gw2 has no compare-routerid statement
Start

r1: router bgp a1 {neighbor n2 remote-as a2}

a1==a2 or a2 a sub-AS?

No

eBGP session to n2 (AS a2)

continue with inter-AS test cases

Yes

Looking for n2 in IGP

r1: router bgp a1 {neighbor n2 next-hop-self}

_ERROR_

r2: interface { ip address n2 }

ERROR: next-hop not in AS

n2 in AS at r2

_ERROR_

router ospf { network [prefix containing n2] }

Next-hop reachability OK

ERROR: next-hop in AS, but not in IGP

_END_
Under the Hood: A Pattern-Based RoLex Rule

Start

r1: router bgp a1 {neighbor n2 remote-as a2}

a1==a2 or a2 a sub-AS?

No

eBGP session to n2 (AS a2)

continue with intra-AS test cases

Yes

ERROR: next-hop not in AS

Looking for n2 in IGP

r1: router bgp a1 {neighbor n2 next-hop-self}

r2: interface { ip address n2 }

_ERROR_

n2 in AS at r2

_ERROR_

Next-hop reachability OK

router ospf { network [prefix containing n2] }

_ERROR_

ERROR: next-hop in AS, but not in IGP
Writing a Pattern-Based RoLex Rule

- RoLex provides finite-state machinery
  - Rules are simple: 41 lines of code for next-hop test

- Rules specify "nodes" and "transitions".

```perl
$no_nh_self_ebgp = sub {
    print STDERR
    $fsm->substitute_def_bindings("ebgp: no next-hop-self _r1 (_n2)\n");
    $fsm->transition(''_r1: router bgp _a1 [[ network _n3 mask _m3
    <contains(_n3/_m3, _n2)>]]',1)->($OK);
    &$ERROR(''_n2 not in iBGP/IGP (eBGP session)'');
};
```

- Network-wide checking is automatic!
Control Flow Analyzer

- Some constraints (e.g., import/export policies) best expressed in terms of higher-level semantics.

- Abstracts mechanisms, gives operators a higher-level view of network configuration.
Control Flow Analyzer: Features

- Graph the network at router-level, labelling route maps on edges.

- Database-backed Web interface.
  - View the number of BGP sessions to each AS.
  - View sessions, import and export route maps:
    - by router
    - associated with a particular remote AS
  - Easily compare policies across routers.

- Policies are "normalized" according to what they do, not what they are called.
Control Flow Analyzer: Network Graph

./cflow.pl --graph=dot,ebgp

- Visualization of import and export policies.
- Routers are nodes, edges are BGP sessions, labels are policies.
- Useful for small networks, sections of larger networks.
Control Flow Analyzer: View by Router

- View all BGP sessions on a particular router.
- Route maps normalized by mechanism.
Control Flow Analyzer: Sessions per AS

- Network-wide view of eBGP and iBGP sessions.
- Can then "drill down" on sessions to a particular AS.

<table>
<thead>
<tr>
<th>Neighbor ASes</th>
<th>Sessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS 209</td>
<td>5</td>
</tr>
<tr>
<td>AS 701</td>
<td>5</td>
</tr>
<tr>
<td>AS 1239</td>
<td>4</td>
</tr>
<tr>
<td>AS 3356</td>
<td>4</td>
</tr>
<tr>
<td>AS 7018</td>
<td>4</td>
</tr>
</tbody>
</table>
Control Flow Analyzer: Sessions per AS

Network-wide view of eBGP and iBGP sessions.

Can then "drill down" on sessions to a particular AS.
Control Flow Analyzer: View By Neighbor AS

<table>
<thead>
<tr>
<th>Router</th>
<th>Neighbor</th>
<th>Neighbor AS</th>
<th>Import Route Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>atlga-gw1</td>
<td>ebgp AS1239 0</td>
<td>1239</td>
<td>25</td>
</tr>
<tr>
<td>cgcil-gw1</td>
<td>ebgp AS1239 1</td>
<td>1239</td>
<td>25</td>
</tr>
<tr>
<td>dlxtx-gw2</td>
<td>ebgp AS1239 2</td>
<td>1239</td>
<td>114</td>
</tr>
<tr>
<td>laxca-gw1</td>
<td>ebgp AS1239 3</td>
<td>1239</td>
<td>25</td>
</tr>
</tbody>
</table>

- Network-wide view of import/export policies to an AS.
- Easy to see when differences exist.
Control Flow Analyzer: View By Neighbor AS

<table>
<thead>
<tr>
<th>Router</th>
<th>Neighbor</th>
<th>Neighbor AS</th>
<th>Import Route Map</th>
<th>Export Route Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>atlga-gw1</td>
<td>ebgp AS1239 0</td>
<td>1239</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>cgcil-gw1</td>
<td>ebgp AS1239 1</td>
<td>1239</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>dlxtx-gw2</td>
<td>ebgp AS1239 2</td>
<td>1239</td>
<td>114</td>
<td>26</td>
</tr>
<tr>
<td>laxca-gw1</td>
<td>ebgp AS1239 3</td>
<td>1239</td>
<td>25</td>
<td>26</td>
</tr>
</tbody>
</table>

- Network-wide view of import/export policies to an AS.
- Easy to see when differences exist.
Control Flow Analyzer: Route Map Diffs

Route Map 25

{ asp(^6451|6451[2-9] )::(^645|645[2-9][0-9] )::(^64|64[6-9][0-9][0-9] )::(^5|65[0-9][0-9][0-9][0-9] )::(()) } { < metric => 0 > < ip => next-hop peer-address > < local-preference => 82 > < community => 0:5000 > }

Route Map 114

{ asp(^6451|6451[2-9] )::(^645|645[2-9][0-9] )::(^64|64[6-9][0-9][0-9] )::(^5|65[0-9][0-9][0-9][0-9] )::(()) } { < metric => 0 > < ip => next-hop peer-address > < local-preference => 82 > < community => 0:5000 > }

Diff Output (zero-indexed)

RoLex: Configuration Verification Suite

- Two distinct tools that parse IOS configs.
  - Pattern-based constraint checker
  - Control flow analyzer
• **Future work:** Check high-level properties.
  - Operator inputs high-level specification
  - High-level network properties checked against constraints

---

**Diagram: RoLex: Configuration Verification Suite**

- **Rules**
- **Cisco IOS**
- **Property Violations**
- **Control Flow Constraint Checker**
- **High-level Network Summary (Web-based interface)**
- **Pattern-Based Constraint Checker**
- **Control Flow Analyzer**
- **High-level Property Specification**

---
Today: Implementing Policy with Mechanism

**Bad:** Import/export route maps, ACLs, communities, etc.

```
neighbor 10.0.0.1 route-map IMPORT-A in
neighbor 10.0.0.1 route-map EXPORT-A out
neighbor 192.168.0.1 route-map IMPORT-C in
neighbor 192.168.0.1 route-map EXPORT-C out
ip community-list 1 permit 0:1000
route-map IMPORT-C permit 10
  set community 0:1000
!
route-map EXPORT-A permit 10
  match community 1
!
```

Diagram:

```
A ---- B ---- C
^         ^
|         |  "Announce p"
|         |
Peer     Peer

```

```
p
```
Ideas for Specifying Information-flow Policy

Better: Lattice model.

Key Challenge: Specification
Control Flow Analyzer: Summary

- Bird’s eye view of network policies.
- Good for spotting anomalies, etc.
- Easy to navigate.

**Other features:**
- View all routers
- Restricted views
  - only eBGP (or iBGP) sessions
  - only import (or export) policies
- Group by common import/export policies

**Coming soon:**
- Specific queries about routes.
- Verify against high-level policy specs (e.g., "lattice").
Towards Intent-based Configuration

Verification requires a specification of intent, which can inspire configuration language design.

- How to specify the information flow lattice?
  - Must be intuitive.
  - Must express varying levels of detail (i.e., AS-level, session-level, prefix-level, etc.)
  - Must express positive requirements, too.

- Expressing intended behavior will improve routing.
  - **Verification**: check existing configurations against intent.
  - **Synthesis**: generate configurations according to intent.
Many Thanks

- Jennifer Rexford
- Randy Bush
Shameless Plea

This tool will only be useful with operator input.

- You need better configuration management tools.
- I need to graduate.

http://nms.lcs.mit.edu/bgp/rolex

- Download the tool, and test it on your configuration.
- Or...I’ll happily test it on your configurations (will write new tests, too).
- Send feedback, feature requests, etc.
Beyond Static Rule Checking

- Statistical inference to reduce manual pain. ("Beliefs")
  - 100 routers, 99 have ACLs configured to deny prefix 192.168.0.0/16.
  - All eBGP sessions to an AS but one have the same import/export policies.

- Capturing dynamic effects. ("Sandbox")
  - Property violations that appear due to timing, message orderings, failures, etc.

- Avoiding low-level silliness. ("Synthesis")
  - Configuration should be specified at the *intent* level, not at the mechanism level.
Verifying Validity

But...low-level checking is not enough.