NetCov: Test Coverage for Network Configurations

Xieyang Xu, Weixin Deng, Ryan Beckett, Ratul Mahajan and David Walker

ANRP Award Talk, IETF120, Jul 22, 2024
Configurations are error-prone

Amazon’s massive AWS outage was caused by human error.

One incorrect command and the whole internet suffers.

By Jason Del Rey | @Dalrey | Mar 2, 2017, 2:20pm EST

Google Cloud Went Down Because It Was Misconfigured

Microsoft: Misconfigured Network Device Caused Azure Outage

Many networks use automated testing to find bugs
Networks fail despite being tested

Facebook outage triggered by BGP configuration issue as services fail for 6 billion
Why would network tests miss the bugs?

- User-provided test suites may be incomplete!
A simple network...
A simple network...

R1's configuration:

```
bgp peer R2
bgp peer ISP
    import policy FROM-ISP

policy FROM-ISP
    match prefix-list INTERNAL
    permit
default
    add tag 74
    permit
...
```

R2's configuration:

```
bgp peer R1
    import policy FROM-R1

policy FROM-R1
    match tag 74
    remove tag 74
    permit
default
deny
...
```
A simple network...

R1’s configuration:

```text
bgp peer R2
bgp peer ISP
    import policy FROM-ISP

policy FROM-ISP
    match prefix-list INTERNAL
    permit
default
    add tag 74
    permit
...
```

R2’s configuration:

```text
bgp peer R1
    import policy FROM-R1

policy FROM-R1
    match tag 74
    remove tag 74
    permit
default
deny
...
```

R1’s routing table

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.00.0.0/8</td>
<td>ISP</td>
<td>74</td>
</tr>
</tbody>
</table>

R2’s routing table

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.00.0.0/8</td>
<td>R1</td>
<td></td>
</tr>
</tbody>
</table>
Test this simple network

R1’s configuration:

```
bgp peer R2
bgp peer ISP
  import policy FROM-ISP

policy FROM-ISP
  match prefix-list INTERNAL
  permit
default
  add tag 74
  permit
...
```

R2’s configuration:

```
bgp peer R1
  import policy FROM-R1

policy FROM-R1
  match tag 74
  remove tag 74
  permit
default
  deny
...
```

R1’s routing table

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.0.0.0/8</td>
<td>ISP</td>
<td>74</td>
</tr>
</tbody>
</table>

R2’s routing table

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.0.0.0/8</td>
<td>R1</td>
<td></td>
</tr>
</tbody>
</table>

Test 1: check configuration contents
R1’s BGP peers include R2 and ISP

Test 2: verify reachability
R2 can reach ISP with any IP in 20/8
Test this simple network

R1’s configuration:

```plaintext
bgp peer R2
bgp peer ISP
  import policy FROM-ISP

policy FROM-ISP
  match prefix-list INTERNAL
  permit

default
  add tag 74
  permit

...```

R2’s configuration:

```plaintext
gbg peer R1
  import policy FROM-R1

policy FROM-R1
  match tag 74
  remove tag 74
  permit

...```

Untested by the test suite. Buggy (supposed to be deny).

R1’s routing table

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.0.0.0/8</td>
<td>ISP</td>
<td>74</td>
</tr>
</tbody>
</table>

R2’s routing table

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.0.0.0/8</td>
<td>R1</td>
<td></td>
</tr>
</tbody>
</table>

Test 1: check configuration contents
R1’s BGP peers include R2 and ISP

Test 2: verify reachability
R2 can reach ISP with any IP in 20/8

Test 3: evaluate routing policy
FROM-ISP should deny internal prefix
What about complete testing of this?
Solution: Guide users with configuration coverage

**Network state**

**R1's configuration:**
```
bgp peer R2
bgp peer ISP
import policy FROM-ISP
policy FROM-ISP
match prefix-list INTERNAL
permit
default
add tag 74
permit
```

**R2's configuration:**
```
bgp peer R1
import policy FROM-R1
policy FROM-R1
match tag 74
remove tag 74
permit
default
deny
```

**R1's routing table**
```
prefix  next hop  tag
20.0.0.0/8  ISP  74
```

**R2's routing table**
```
prefix  next hop  tag
20.0.0.0/8  R1  74
```

**Test suite**

**Test 1: check configuration content**

**Test 2: verify reachability**

**Which configuration lines are tested, and which ones are not?**

Not tested
Defining configuration coverage

- Lines *directly analyzed* by tests are covered.

**R1's configuration:**
```
bgp peer R2
bgp peer ISP
  import policy FROM-ISP
policy FROM-ISP
  match prefix-list INTERNAL
  default
    permit
    add tag 74
    permit
  ...
```

**R2's configuration:**
```
bgp peer R1
  import policy FROM-R1
policy FROM-R1
  match tag 74
  remove tag 74
  permit
  default
  deny
  ...
```

**R1's routing table**
```
prefix   | next hop | tag |
---------|----------|-----|
20.0.0.0/8 | ISP     | 74  |
```

**R2's routing table**
```
prefix   | next hop | tag |
---------|----------|-----|
20.0.0.0/8 | R1      |    |
```

Test 1: check configuration contents
R1's BGP peers include R2 and ISP

Test 2: verify reachability
R2 can reach ISP with any IP in 20/8
Defining configuration coverage

▷ Lines *directly analyzed* by tests are covered.

▷ Lines *contribute to* tested data plane states are covered.

Test 1: check configuration contents
R1's BGP peers include R2 and ISP

Test 2: verify reachability
R2 can reach ISP with any IP in 20/8
Defining configuration coverage

▷ Lines *directly analyzed* by tests are covered.
▷ Lines *contribute to* tested data plane states are covered.
  ○ Contributors: critical to the existence, local or non-local.
Key problem

Goal: efficiently map data plane states back to contributors.

Strawman solutions:

1. Full data plane simulation and record the contributions at each step.
2. Encode control plane computation as deductive clauses, which can be used to infer contributions on demand[^1].

Insight

▷ The network state itself often hints the contributors!

RIB entry @R2
20.0.0.0/8 -> R1 (BGP)

BGP peer config @R1
bgp peer R2

BGP peer config @R2
bgp peer R1

Import policy @R2
match tag 74
permit

BGP session
R1 ↔ R2

BGP announcement
R1 → R2, 20.0.0.0/8

RIB entry @R1
20.0.0.0/8 -> ISP (BGP)
Approach overview

▷ Information flow model: a graph model of network contributions.
▷ Infer contributions on demand with heuristics and local simulations.

https://github.com/UWNetworksLab/netcov
NetCov design

Configurations → NetCov → Configuration Coverage

Data plane state* → NetCov

Test trace** → NetCov

*Retrieved from live networks or simulated/emulated.
**Directly analyzed configurations lines and tested data plane state entries.
LCOV - code coverage report

Current view: top level
Test: internet2.initial-tests
Date: 2022-09-20 14:54:06

<table>
<thead>
<tr>
<th>Directory</th>
<th>Line Coverage</th>
<th>Hit</th>
<th>Total</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>config</td>
<td>26.1 %</td>
<td>16912</td>
<td>64886</td>
<td>26.1 %</td>
</tr>
</tbody>
</table>

Generated by: LCOV version 1.15

`Live Demo`
Case study: Internet2

- 10 BGP routers
- Over 90,000 lines of configuration
- Use Route Views[2] dataset for route announcements from 268 external peers

Existing test suite

- Bagpipe[3] verified Internet2 BGP configuration with 3 tests:
  - Block-to-external
  - No Martian
  - Route preference

Coverage results of existing tests

The tests left most of the configurations untested.

Fraction of configuration lines covered:

<table>
<thead>
<tr>
<th>Block-To-External</th>
<th>No-Martian</th>
<th>Route-Preference</th>
<th>Test suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6%</td>
<td>0.9%</td>
<td>24%</td>
<td>26%</td>
</tr>
</tbody>
</table>
Improve tests with NetCov

- **NoMartian** only covers one of five terms of the import policy.
- 4 other classes of forbidden traffic remain untested.
- We add a new test checking that Internet2 should reject these traffic.
- **Policy SANITY-IN** get fully covered.
Coverage was improved effectively

From 26% to 43% after 3 iterations

Fraction of configuration lines covered:

- Original tests: 26%
- Add Sanity-In: 27%
- Add Peer-Specific-Route: 37%
- Add Interface-Reachability: 43%
Conclusion

▷ Complete testing is hard by users alone.
▷ We define and compute configuration coverage.
  ○ Key problem: efficiently map data plane states back to contributors.
  ○ Our approach: information-flow model and on-demand inference.
▷ With NetCov, we can make network tests more complete.

https://github.com/UWNetworksLab/netcov

pip install netcov