NetCov: Test Coverage for Network Configurations

Xieyang Xu, Weixin Deng, Ryan Beckett, Ratul Mahajan and David Walker
Because network configuration is error-prone ...

... many networks use automatic testing to reduce risk
But networks fail despite automatic testing (because of bugs that testing should have caught!)

Facebook outage triggered by BGP configuration issue as services fail for 6 billion

WAN router IP address change blamed for global Microsoft 365 outage

Command line not vetted using full qualification process, says Redmond. We think it involved chewing gum somewhere
But networks fail despite automatic testing (because of bugs that testing should have caught!)
An example of a testing gap
An example of a testing gap

R1’s configuration:

```config
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:

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

policy FROM-R1
  match tag 74
  remove tag 74
  permit
  default
  deny
...
```
An example of a testing gap

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

...```

R1’s routing table

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.00.0/8</td>
<td>ISP</td>
<td>74</td>
</tr>
</tbody>
</table>

R2’s configuration:

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

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

...```

R2’s routing table

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.00.0/8</td>
<td>R1</td>
<td></td>
</tr>
</tbody>
</table>
An example of a testing gap

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
An example of a testing gap

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
```

Undetected bug! (should be deny).

R2's configuration:

```
bgp peer R1
   import policy FROM-R1

policy FROM-R1
   match tag 74
   remove tag 74
   permit
```

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

R1’s routing table:

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.00.0/8</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>20.00.0/8</td>
<td>R1</td>
<td></td>
</tr>
</tbody>
</table>
What about complete testing of this?
Solution: Guide users with configuration coverage

Network state

Test suite

Test 1: check configuration content

Test 2: verify reachability

New test: evaluate routing policy

User

Which config lines are tested?

Not tested

R1's configuration:
```
bgp peer R2
bgp peer ISP
import policy FROM-ISP
policy FROM-ISP
    match prefix-list INTERNAL
    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
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
```

prefix  next hop  tag
20.0.0.0/8  ISP  74

prefix  next hop  tag
20.0.0.0/8  R1  74

Not tested

prefix  next hop  tag
20.0.0.0/8  ISP  74

prefix  next hop  tag
20.0.0.0/8  R1  74

prefix  next hop  tag
20.0.0.0/8  ISP  74

prefix  next hop  tag
20.0.0.0/8  R1  74
Defining configuration coverage

1. Lines that are *directly analyzed* by tests

---

**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
Defining configuration coverage

1. Lines that are *directly analyzed* by tests
2. Lines that *contribute to* tested data plane 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

<table>
<thead>
<tr>
<th>prefix</th>
<th>next hop</th>
<th>tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.00.0/8</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>2000.0/8</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
Defining configuration coverage

1. Lines that are *directly analyzed* by tests
2. Lines that *contribute to* tested data plane state
   - Can be non-local
Key challenge

- Efficiently mapping 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
Key insight

▷ The network state (often) contains hints to infer contributors!
Solution overview

▷ Information flow graph to model network contributions
  ○ Encoded as rules that derive ancestors of nodes
  ○ Sometimes derivations need local simulations

▷ Infers contributions on demand
  ○ Repeatedly run the rules to fixed point

▷ Accounts for non-determinism
  ○ BGP aggregates, multipath routing
NetCov design

- Configurations
- Data plane state
- Test trace

Directly analyzed config lines, tested data plane state

NetCov

Configuration Coverage
Demo
Case study: Internet2

▷ 10 BGP routers
▷ 90K+ lines of configuration
▷ 268 external peers
▷ Use RouteViews data to infer external route announcements
Existing test suite

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

Existing test suite covered only 1 in 4 lines

Fraction of configuration lines covered

- Block-To-External: 0.6%
- No-Martian: 0.9%
- Route-Preference: 24%
- Test suite: 26%
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.
New tests significantly improved coverage

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original test suite</td>
<td>26%</td>
</tr>
<tr>
<td>Add Sanity-In</td>
<td>27%</td>
</tr>
<tr>
<td>Add Peer-Specific-Route</td>
<td>37%</td>
</tr>
<tr>
<td>Add Iface-Reachability</td>
<td>43%</td>
</tr>
</tbody>
</table>
Coverage can be computed in reasonable time
Conclusion

▷ Need high-quality test suites to make networks reliable
  ○ Simply using automated testing is insufficient

▷ NetCov improves test suites by revealing test coverage of configs
  ○ Key challenge: map data plane state back to contributors
  ○ Our approach: information-flow model and on-demand inference

https://github.com/UWNetworksLab/netcov