NetCov: Test Coverage for Network Configurations

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

UNIVERSITY of WASHINGTON Microsoft PRINCETON UNIVERSITY

Because network configuration is error-prone …

Amazon's massive AWS outage was caused by human error

One incorrect command and the whole internet suffers.

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

Google Cloud Went Down Because It Was Misconfigured

on JUNE 7, 2019 Written by Bill Hartzer

Microsoft: Misconfigured Network Device Caused Azure Outage

… many networks use automatic testing to reduce risk

Reachability Analysis for AWS-based Networks

Accuracy, Scalability, Coverage – A Practical Configuration Verifier on a Global WAN

Fangdan Ye[⋆△‡], Da Yu[§△‡], Ennan Zhai[†], Hongqiang Harry Liu[†], Bingchu
Chunsheng Wang[†], Xin Wu[†], Tianchen Guo[†], Cheng Jin[†], Duncher
Biao Cheng[†], Hui Xu[†], Ming Zhang[†□], Zhiliang Wang[⋆□], Rod
[†]*Alibaba Group* [⋆]*Tsinghua University* [§]*Brown University*

[2], C. Dodge[1], A. Gacek[1], A.J. Hu[4], T.

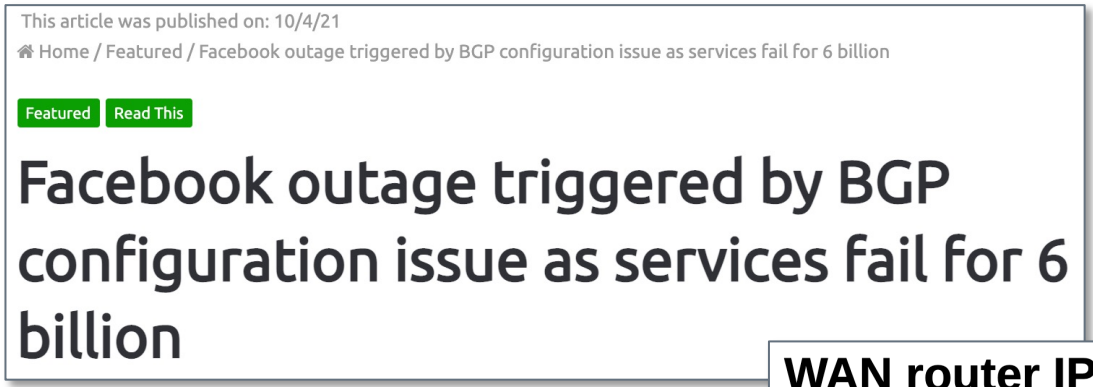
Validating Datacenters At Scale

Karthick Jayaraman, Nikolaj Bjørner, Jitu Padhye, Amar Agrawal, Ashish Bhargava, Paul-Andre C Bissonnette, Shane Foster, Andrew Helwer, Mark Kasten, Ivan Lee, Anup Namdhari, Haseeb Niaz, Aniruddha Parkhi, Hanukumar Pinnamraju, Adrian Power, Neha Milind Raje, Parag Sharma

Microsoft

networkverification@microsoft.com

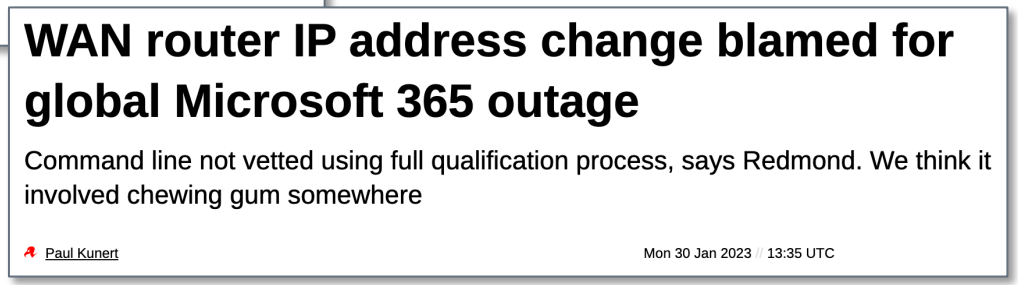
But networks fail despite automatic testing (because of bugs that testing should have caught!)


This article was published on: 10/4/21

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

Featured Read This

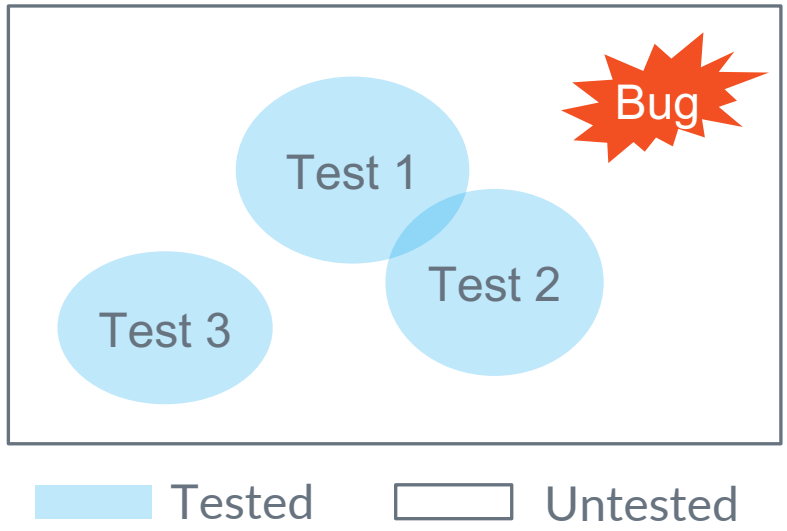
Facebook outage triggered by BGP configuration issue as services fail for 6 billion


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

Paul Kunert Mon 30 Jan 2023 13:35 UTC

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:

```
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
…
```

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

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	

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
…
```

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

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	

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
…
```

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

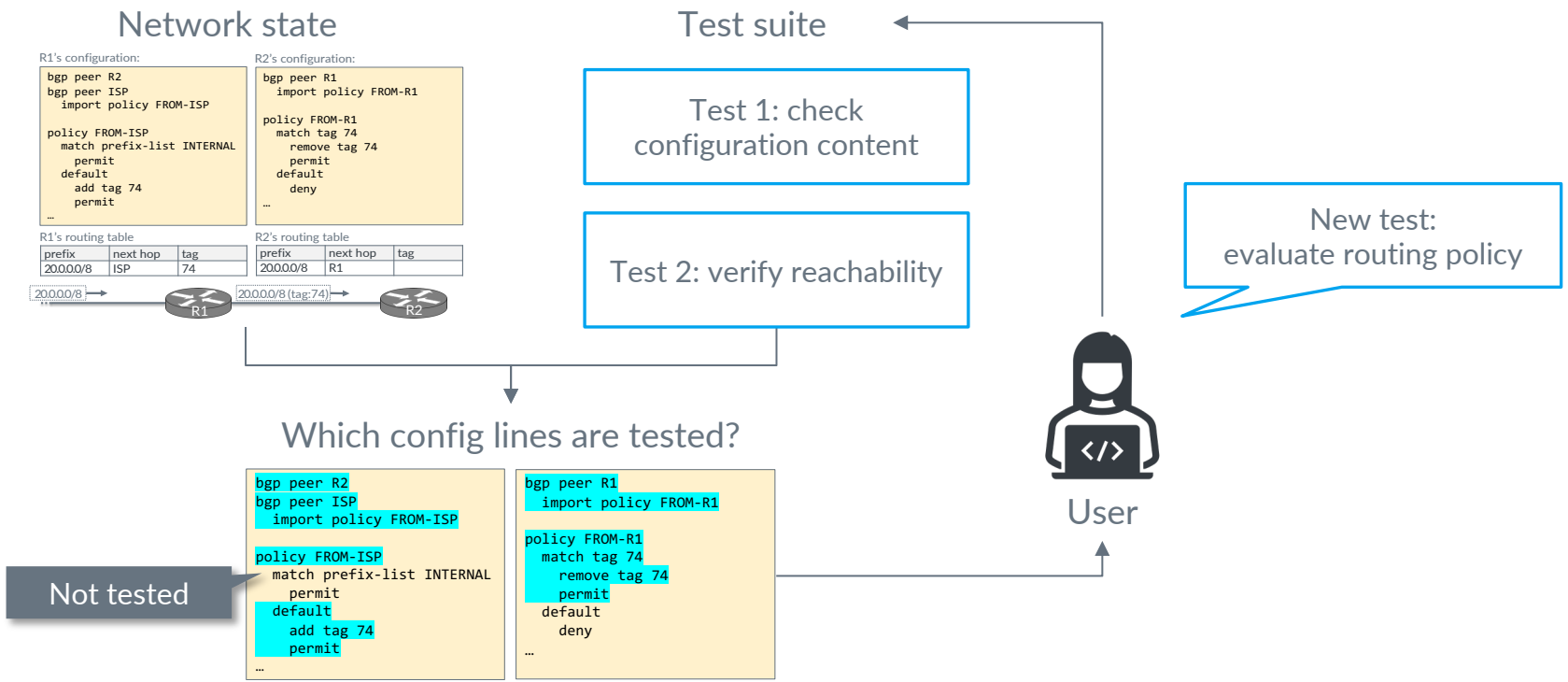
Test 2: verify reachability
R2 can reach ISP with any IP in 20/8

Test 3: evaluate routing policy
FROM-ISP should deny internal prefix

What about complete testing of this?

Credit: Microsoft

Solution: Guide users with configuration coverage



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
…
```

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

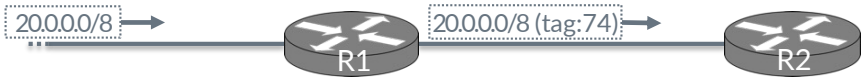
Test 2: verify reachability
R2 can reach ISP with any IP in 20/8

R1's routing table

prefix	next hop	tag
20.0.0.0/8	ISP	74

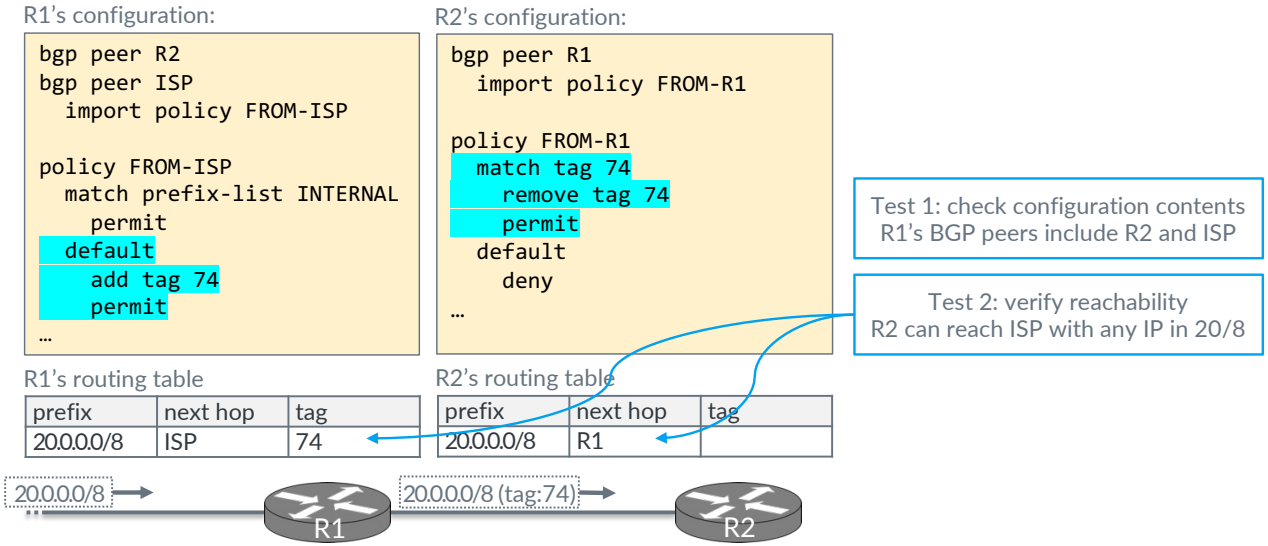
R2's routing table

prefix	next hop	tag
20.0.0.0/8	R1	



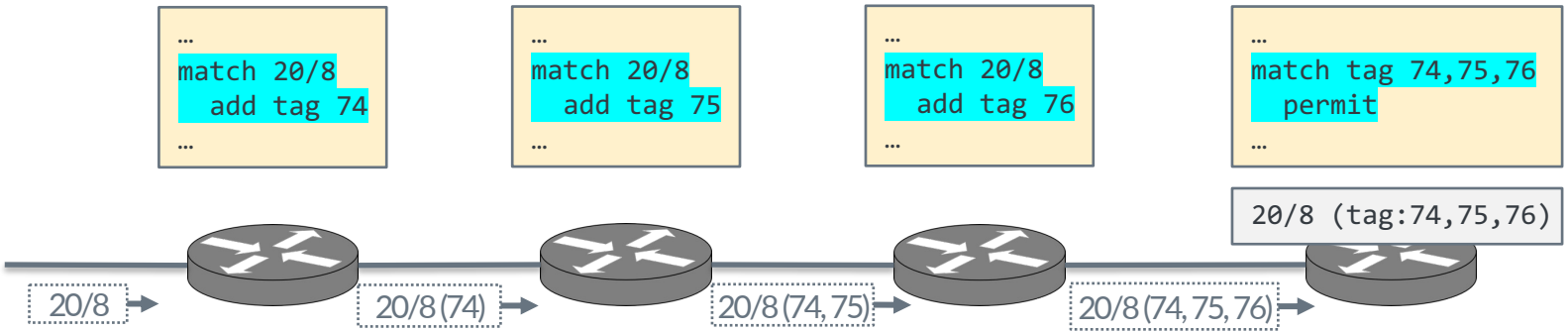
Defining configuration coverage

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



Defining configuration coverage

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



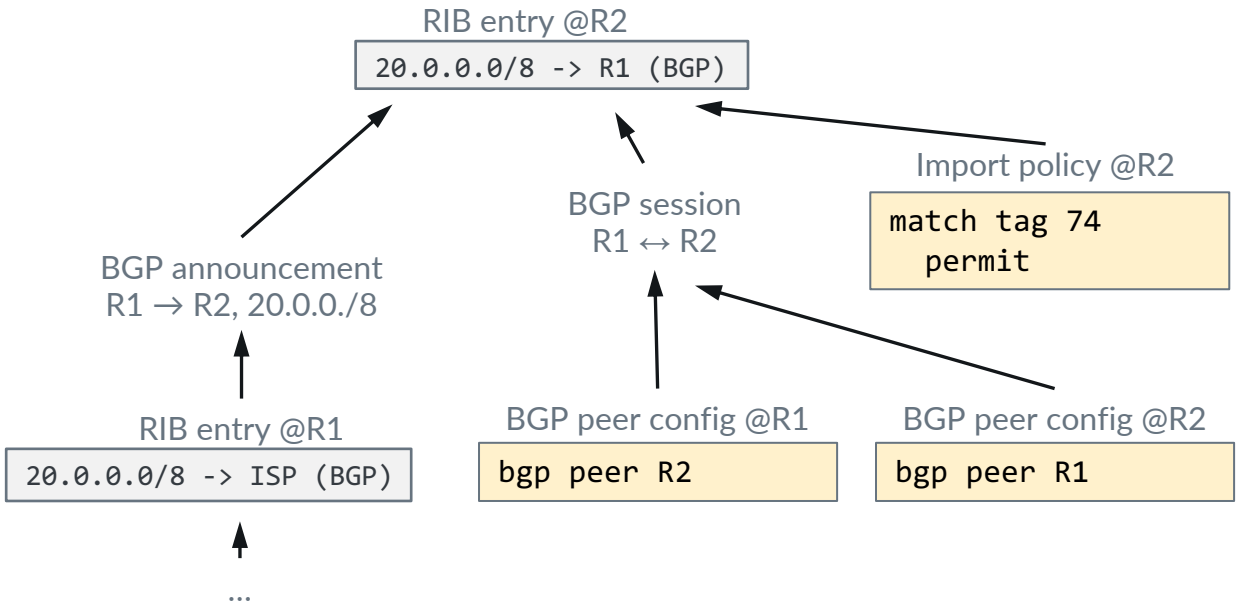
Key challenge

- 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

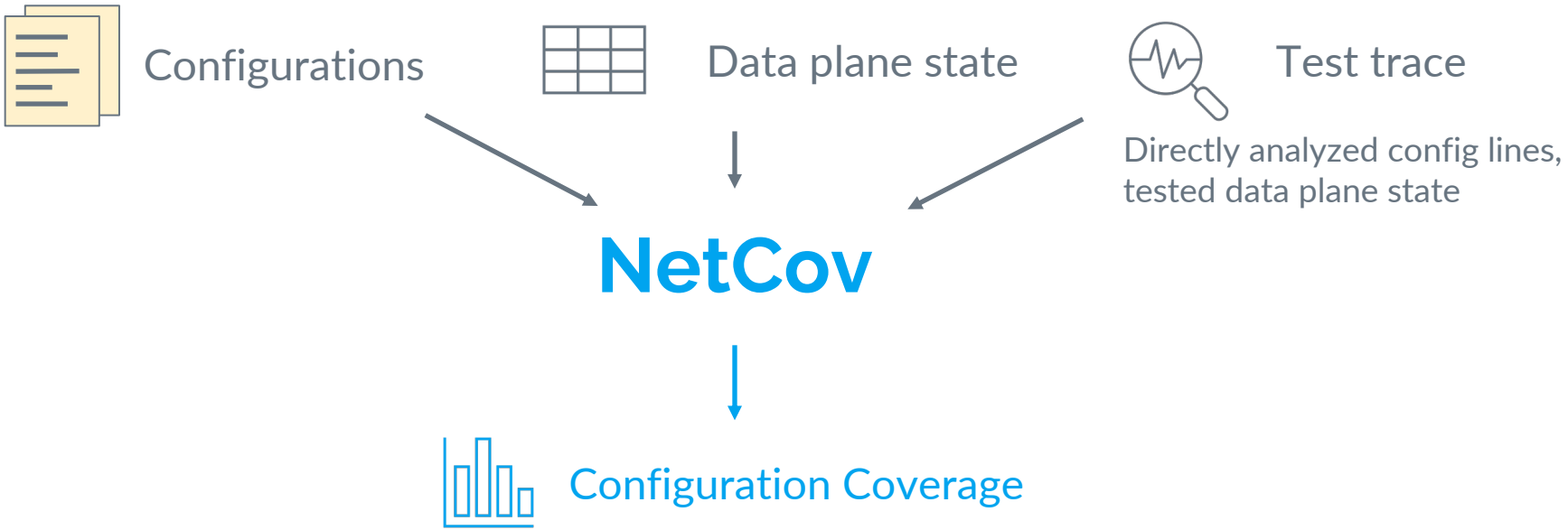
- The network state (often) contains hints to infer contributors!



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



Demo

Case study: Internet2

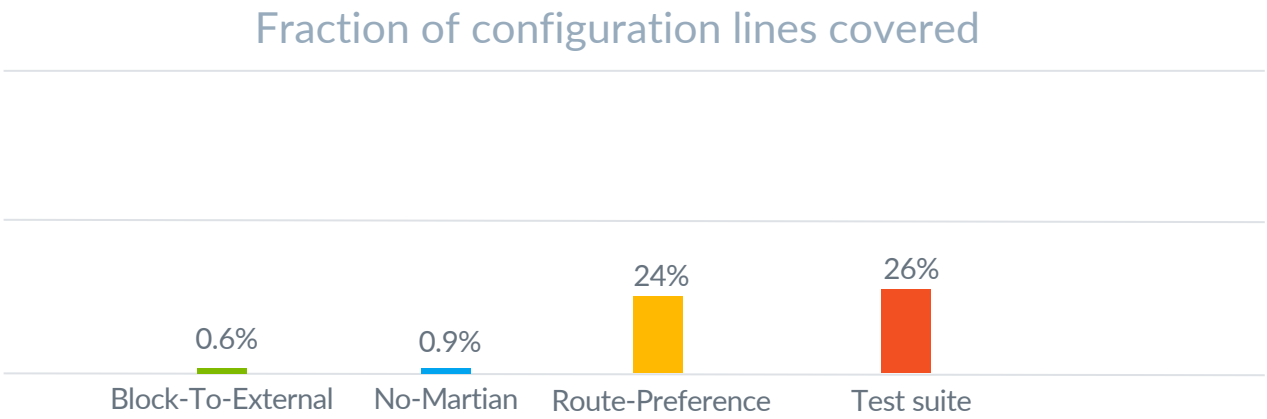
- 10 BGP routers
- 90K+ lines of configuration
- 268 external peers
- Use RouteViews data to infer external route announcements

Existing test suite

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

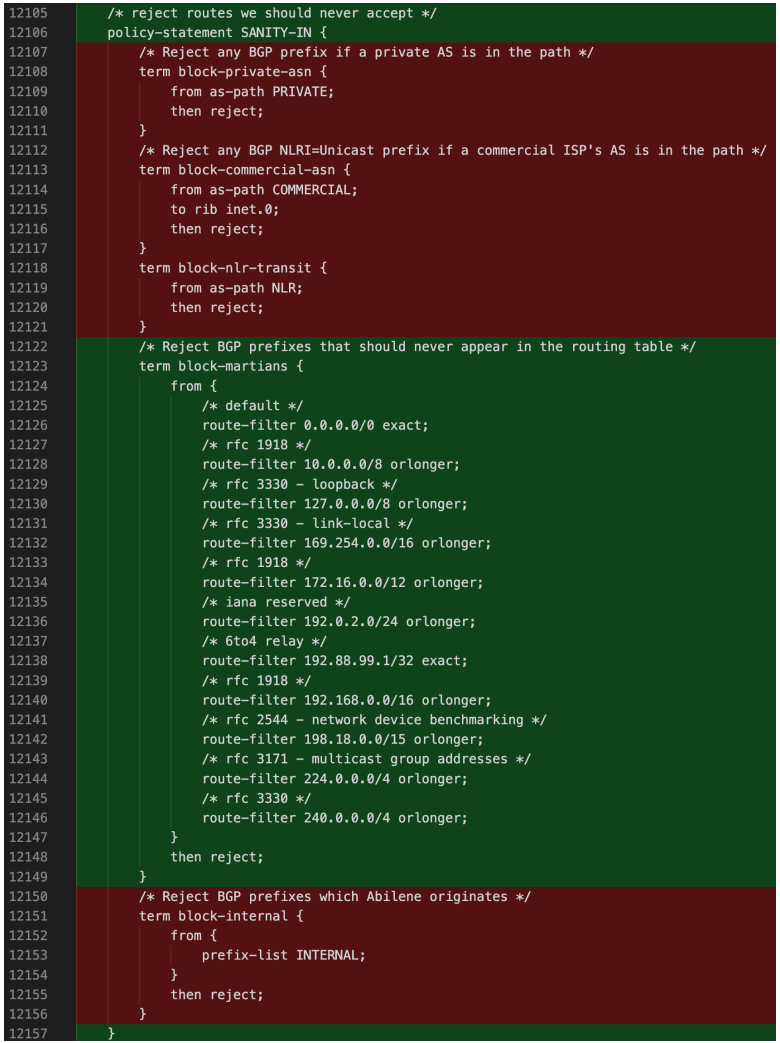
*Weitz et al. *Scalable verification of border gateway protocol configurations with an SMT solver.* In *OOPSLA 2016.*

Existing test suite covered only 1 in 4 lines



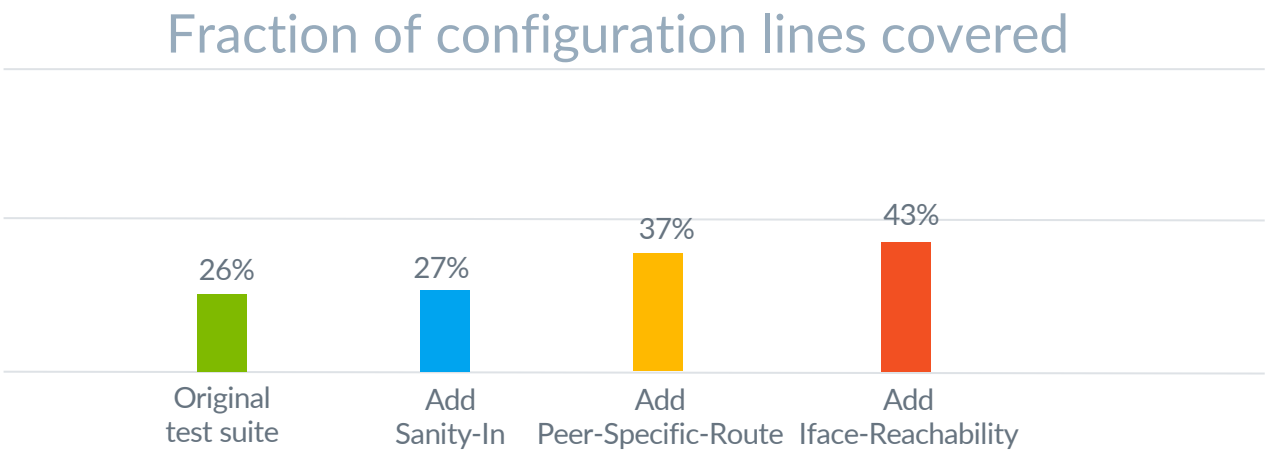
Improve tests with NetCov

- *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.

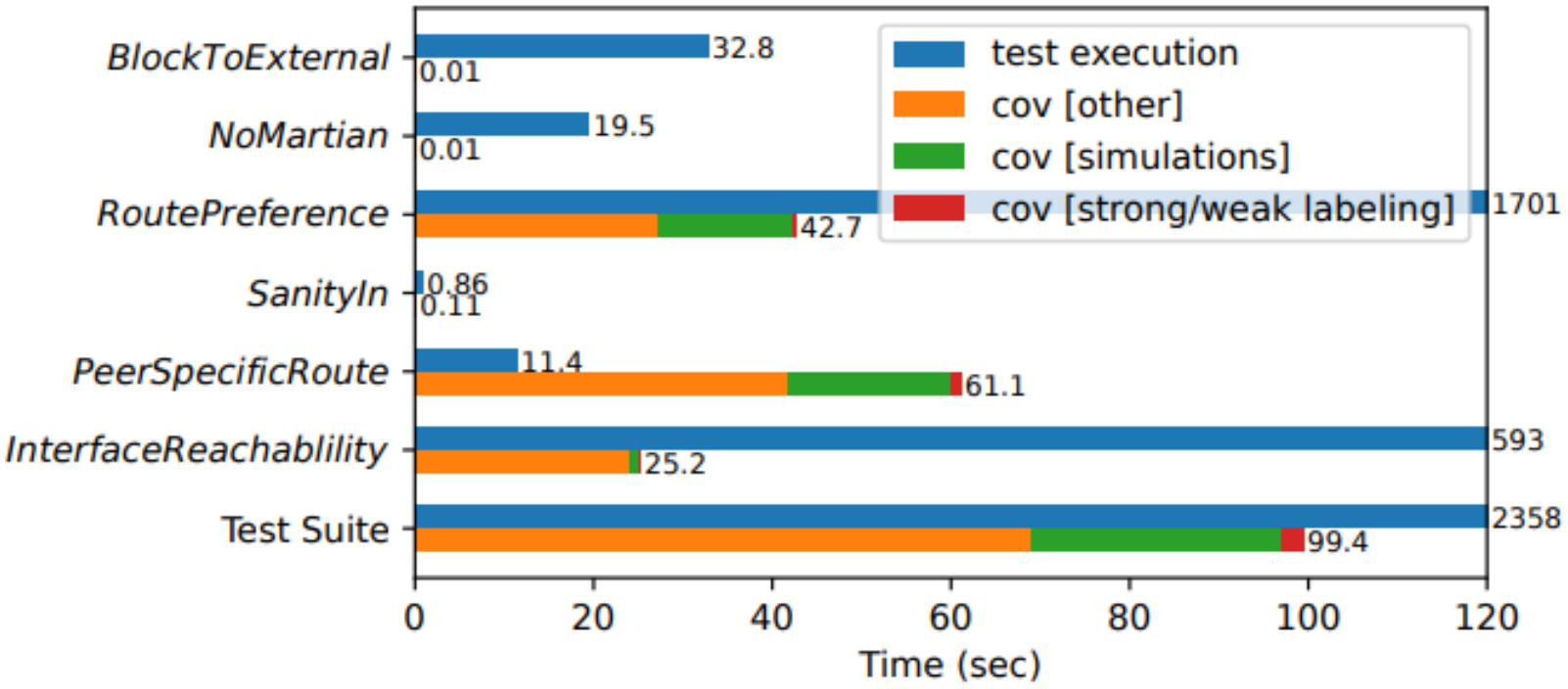


```
12105     /* reject routes we should never accept */
12106     policy-statement SANITY-IN {
12107         /* Reject any BGP prefix if a private AS is in the path */
12108         term block-private-asn {
12109             from as-path PRIVATE;
12110             then reject;
12111         }
12112         /* Reject any BGP NLRI=Unicast prefix if a commercial ISP's AS is in the path */
12113         term block-commercial-asn {
12114             from as-path COMMERCIAL;
12115             to rib inet.0;
12116             then reject;
12117         }
12118         term block-nlr-transit {
12119             from as-path NLR;
12120             then reject;
12121         }
12122         /* Reject BGP prefixes that should never appear in the routing table */
12123         term block-martians {
12124             from {
12125                 /* default */
12126                 route-filter 0.0.0.0/0 exact;
12127                 /* rfc 1918 */
12128                 route-filter 10.0.0.0/8 orlonger;
12129                 /* rfc 3330 - loopback */
12130                 route-filter 127.0.0.0/8 orlonger;
12131                 /* rfc 3330 - link-local */
12132                 route-filter 169.254.0.0/16 orlonger;
12133                 /* rfc 1918 */
12134                 route-filter 172.16.0.0/12 orlonger;
12135                 /* iana reserved */
12136                 route-filter 192.0.2.0/24 orlonger;
12137                 /* 6to4 relay */
12138                 route-filter 192.88.99.1/32 exact;
12139                 /* rfc 1918 */
12140                 route-filter 192.168.0.0/16 orlonger;
12141                 /* rfc 2544 - network device benchmarking */
12142                 route-filter 198.18.0.0/15 orlonger;
12143                 /* rfc 3171 - multicast group addresses */
12144                 route-filter 224.0.0.0/4 orlonger;
12145                 /* rfc 3330 */
12146                 route-filter 240.0.0.0/4 orlonger;
12147             }
12148             then reject;
12149         }
12150         /* Reject BGP prefixes which Abilene originates */
12151         term block-internal {
12152             from {
12153                 prefix-list INTERNAL;
12154             }
12155             then reject;
12156         }
12157     }
```

New tests significantly improved coverage



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`