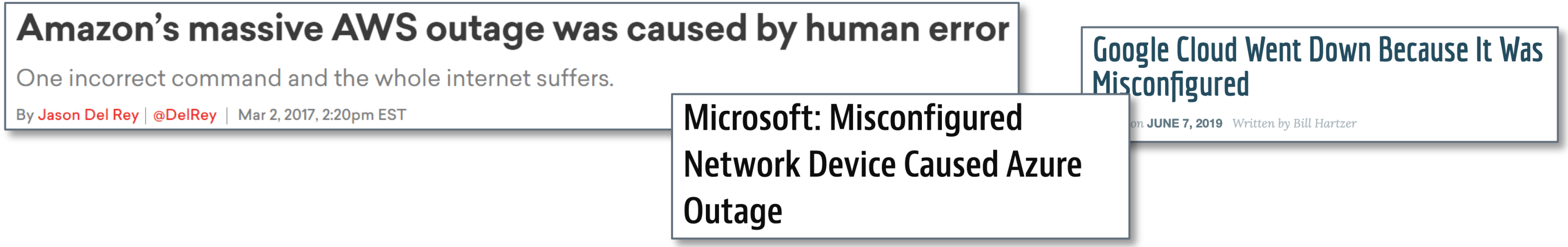
NetCov: Test Coverage for Network Configurations

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

ANRP Award Talk, IETF120, Jul 22, 2024

UNIVERSITY of WASHINGTON Microsoft PRINCETON UNIVERSITY

Configurations are error-prone



Amazon's massive AWS outage was caused by human error

One incorrect command and the whole internet suffers.

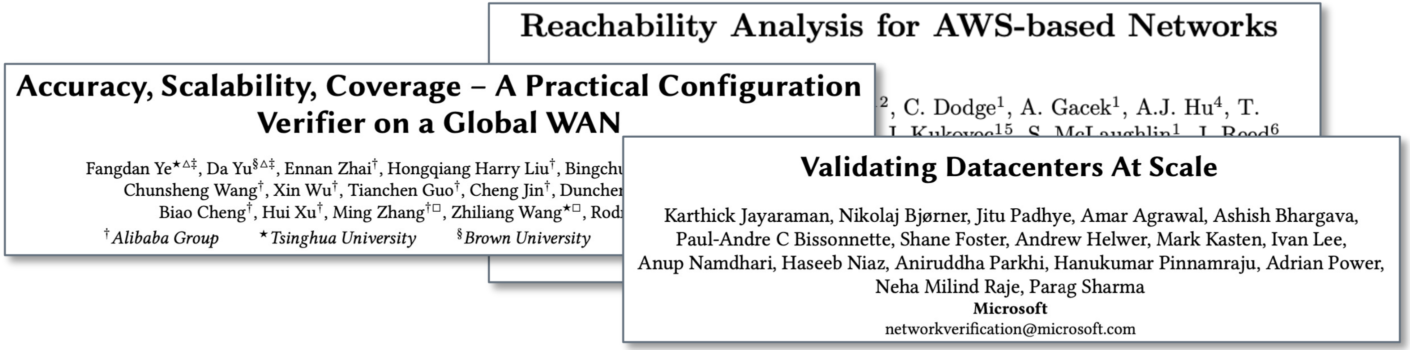
By Jason Del Rey | @DelRey | Mar 2, 2017, 2:20pm EST

Microsoft: Misconfigured Network Device Caused Azure Outage

Google Cloud Went Down Because It Was Misconfigured

on JUNE 7, 2019 *Written by Bill Hartzer*

Many networks use automated testing to find bugs



Reachability Analysis for AWS-based Networks

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

J. Kukovec[1,5], S. McLaughlin[1], J. Reed[6]

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

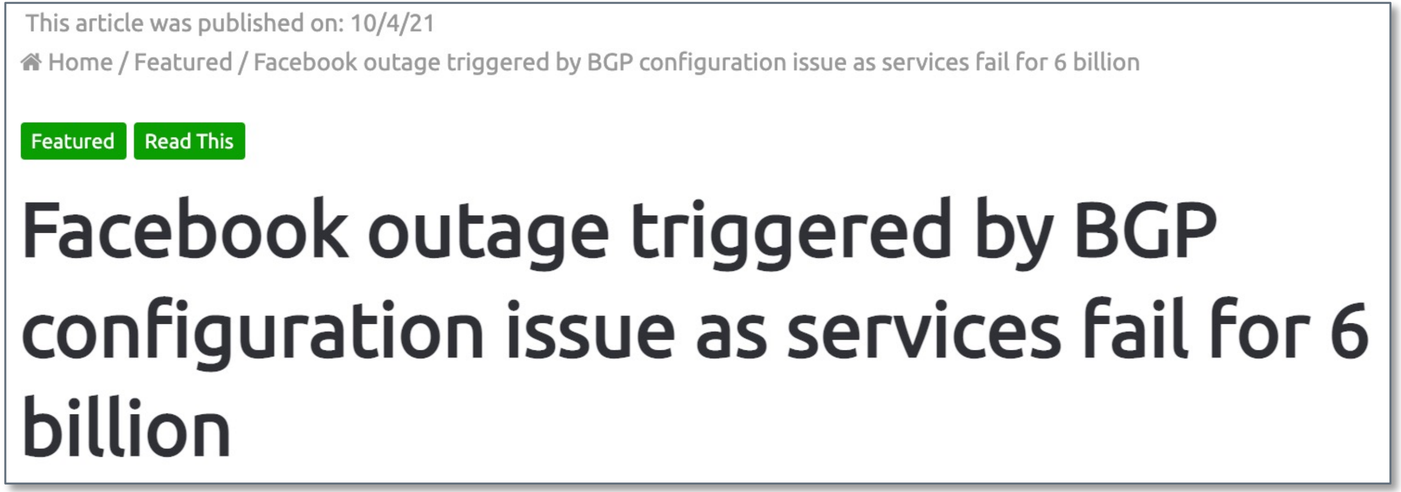
Fangdan Ye[⋆△‡], Da Yu[§△‡], Ennan Zhai[†], Hongqiang Harry Liu[†], Bingchu
Chunsheng Wang[†], Xin Wu[†], Tianchen Guo[†], Cheng Jin[†], Dunchen
Biao Cheng[†], Hui Xu[†], Ming Zhang[†□], Zhiliang Wang[⋆□], Rodr

[†] *Alibaba Group* [⋆] *Tsinghua University* [§] *Brown University*

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

Networks fail despite being tested



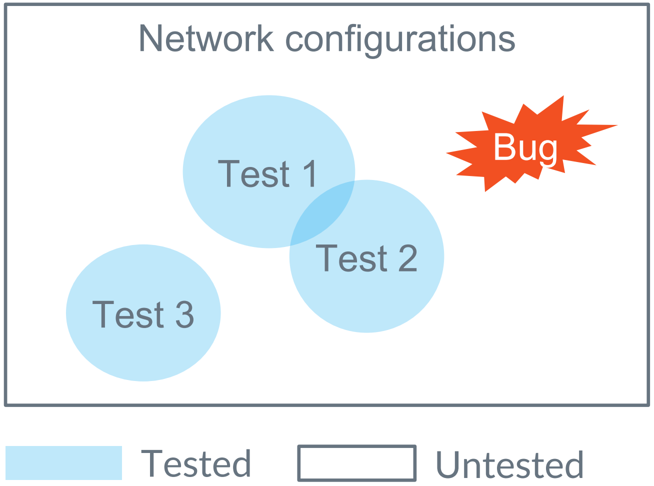
This article was published on: 10/4/21

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

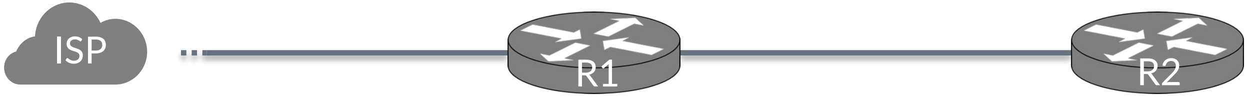
Facebook outage triggered by BGP configuration issue as services fail for 6 billion

Why would network tests miss the bugs?

- User-provided test suites may be incomplete!



A simple network…



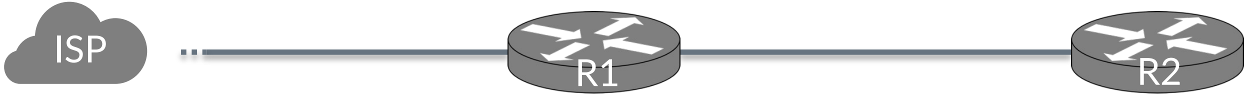
A simple network…

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:

```
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:

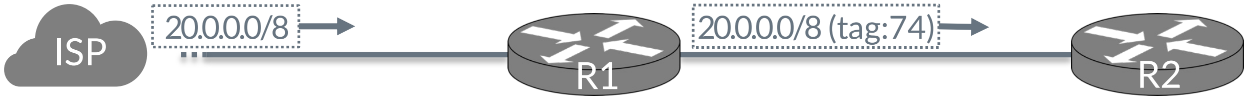
```
bgp peer R1
  import policy FROM-R1

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	



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
…
```

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

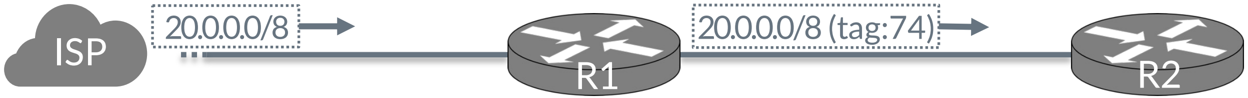
Test 2: verify reachability
R2 can reach ISP with any IP in 20/8

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	



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
…
```

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

R2's configuration:

```
bgp peer R1
  import policy FROM-R1

policy FROM-R1
  match tag 74
    remove tag 74

…
```

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

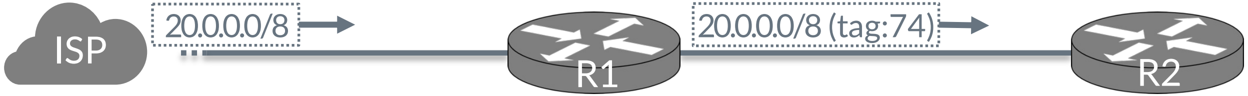
Test 3: evaluate routing policy
FROM-ISP should deny internal prefix

R1's routing table

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	

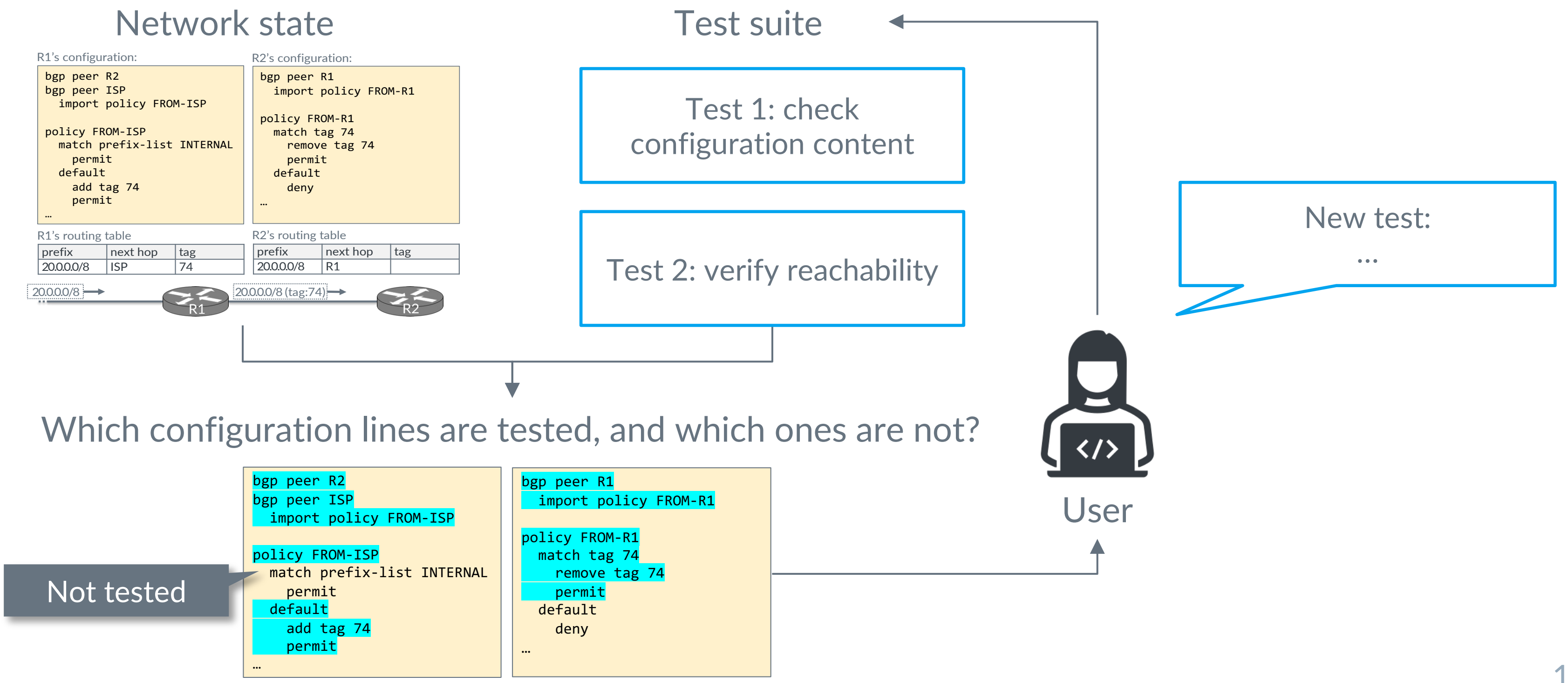


What about complete testing of this?

● Datacenter
● Edge
Network

Credit: Microsoft

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	

Test suite

Test 1: check configuration content

Test 2: verify reachability

New test:
…

User

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

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

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

```
bgp peer R1
  import policy FROM-R1

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

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
    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

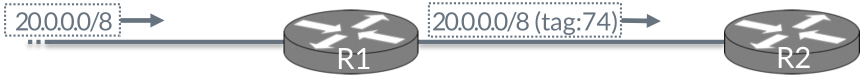
Test 2: verify reachability
R2 can reach ISP with any IP in 20/8

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

- ▷ Lines *directly analyzed* by tests are covered.
- ▷ Lines *contribute to* tested data plane states are covered.

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

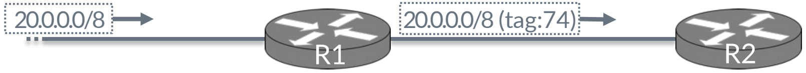
Test 2: verify reachability
R2 can reach ISP with any IP in 20/8

R1's routing table

prefix	next hop	tag
20.0.0.0/8	ISP	74

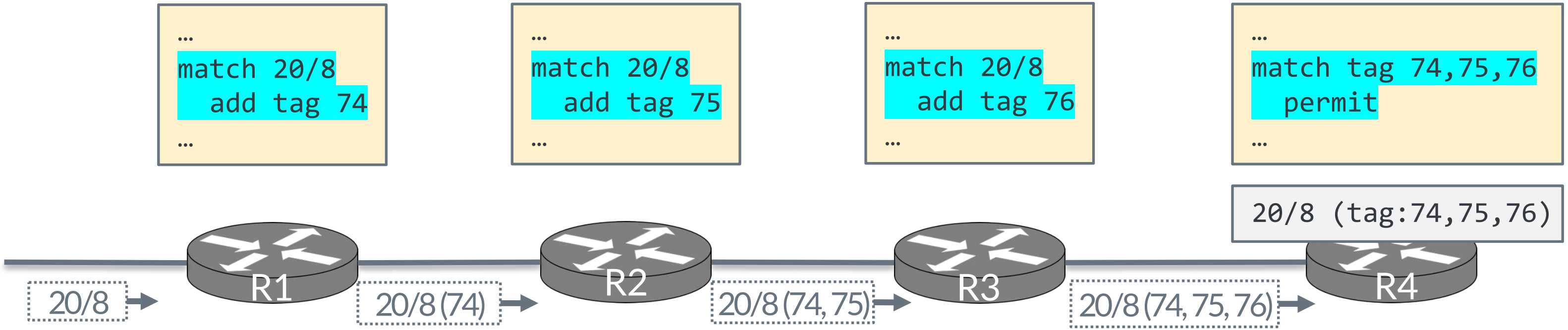
R2's routing table

prefix	next hop	tag
20.0.0.0/8	R1	



Defining configuration coverage

- 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.



```
…
match 20/8
  add tag 74
…
```

```
…
match 20/8
  add tag 75
…
```

```
…
match 20/8
  add tag 76
…
```

```
…
match tag 74,75,76
  permit
…
```

```
20/8 (tag:74,75,76)
```

Key problem

▷ Goal: efficiently map data plane states back to contributors.

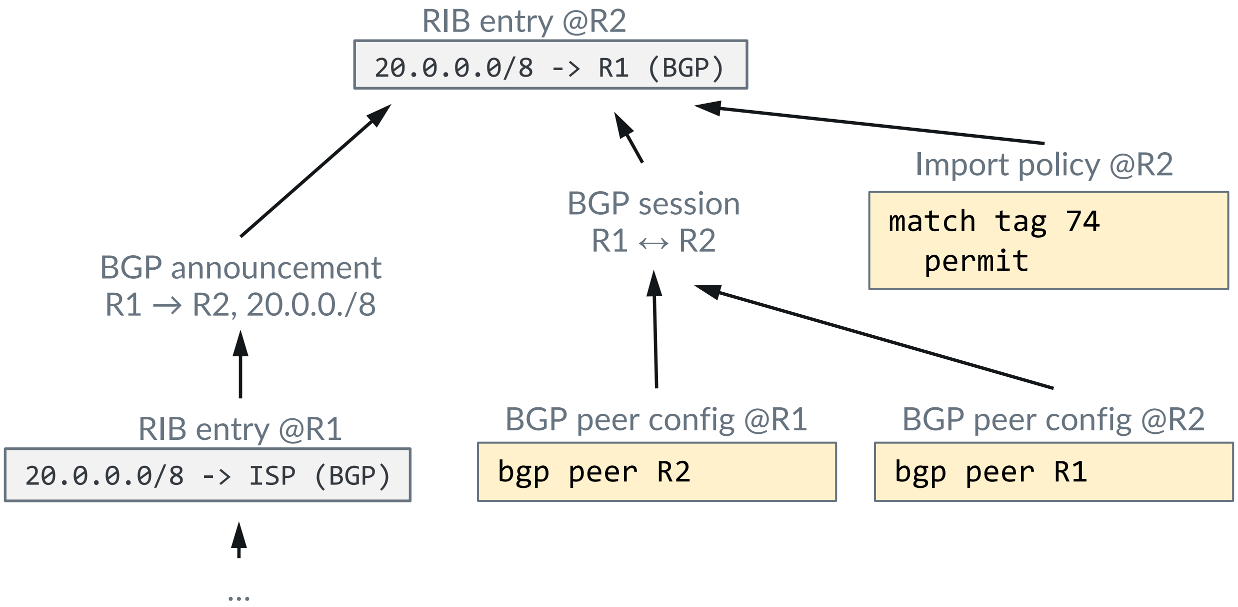
▷ Strawman solutions:

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].

[1] Zhou et al. *Efficient Querying and Maintenance of Network Provenance at Internet-Scale.* In *SIGMOD 2010.*

Insight

- The network state itself often hints the contributors!


RIB entry @R2

```
20.0.0.0/8 -> R1 (BGP)
```

Import policy @R2

```
match tag 74
  permit
```

BGP session
R1 ↔ R2

BGP announcement
R1 → R2, 20.0.0./8

RIB entry @R1

```
20.0.0.0/8 -> ISP (BGP)
```

...

BGP peer config @R1

```
bgp peer R2
```

BGP peer config @R2

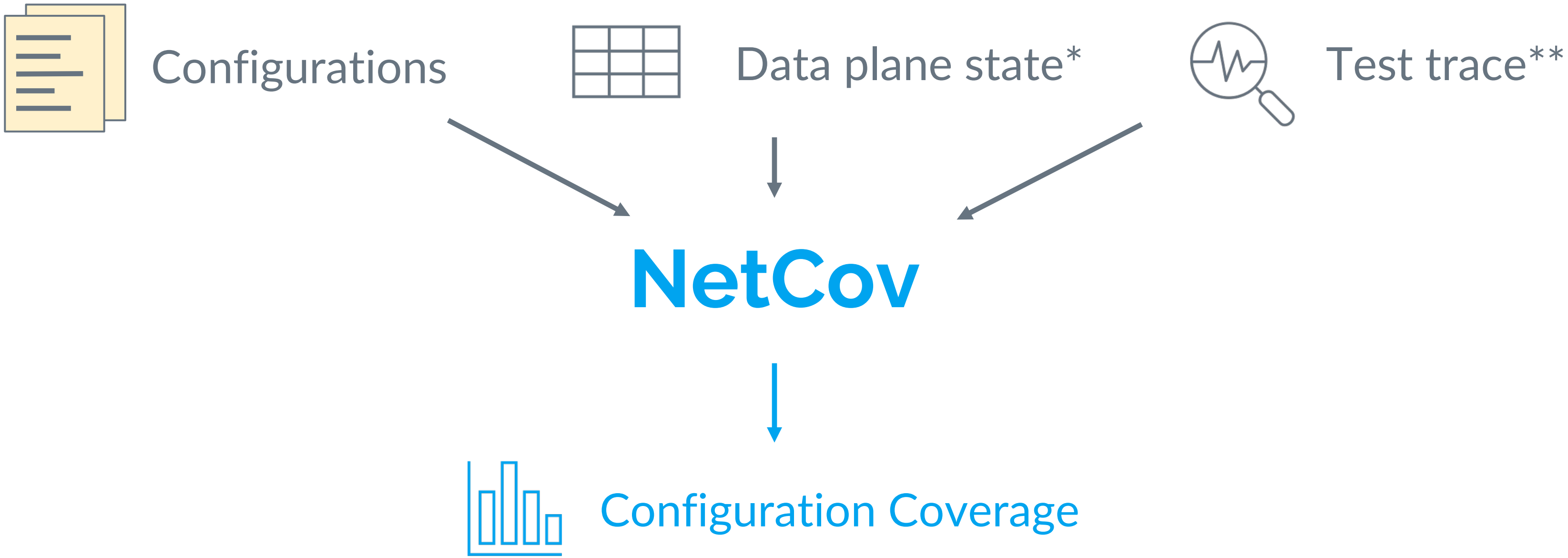
```
bgp peer R1
```

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

Data plane state*

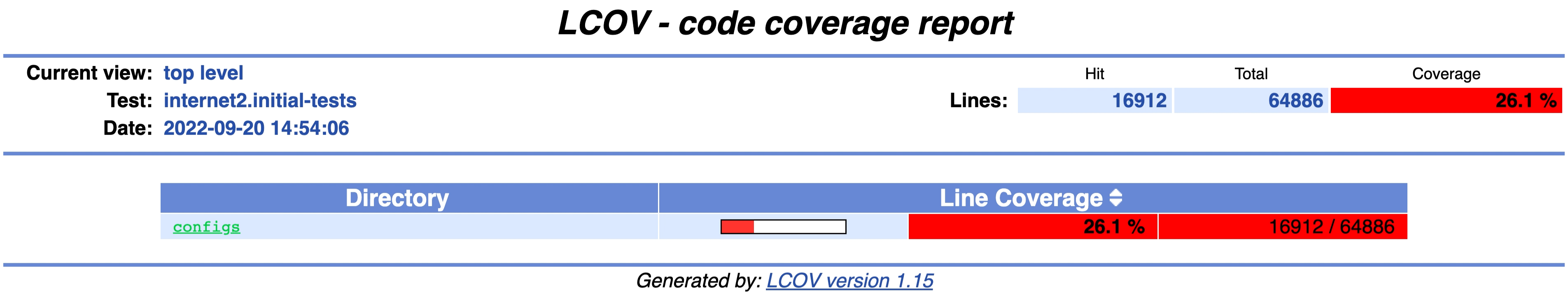
Test trace**

NetCov

Configuration Coverage

*Retrieved from live networks or simulated/emulated.

**Directly analyzed configurations lines and tested data plane state entries.



LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level				
Test:	internet2.initial-tests	**Lines:**	16912	64886	26.1 %
Date:	2022-09-20 14:54:06				

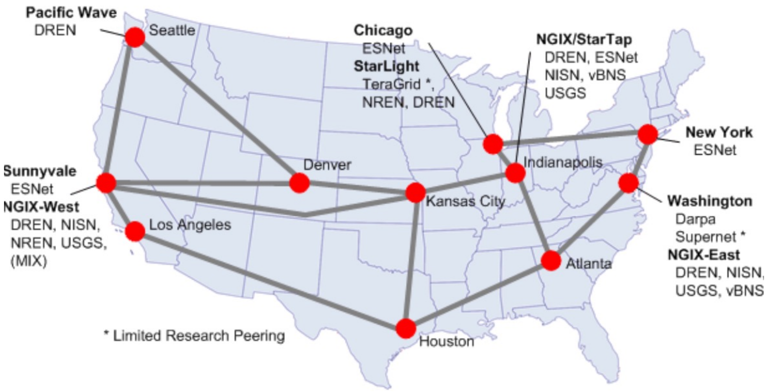
Directory	Line Coverage ⇕		
configs		26.1 %	16912 / 64886

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



[2] *University of Oregon Route Views Project*. https://www.routeviews.org/routeviews/

Existing test suite

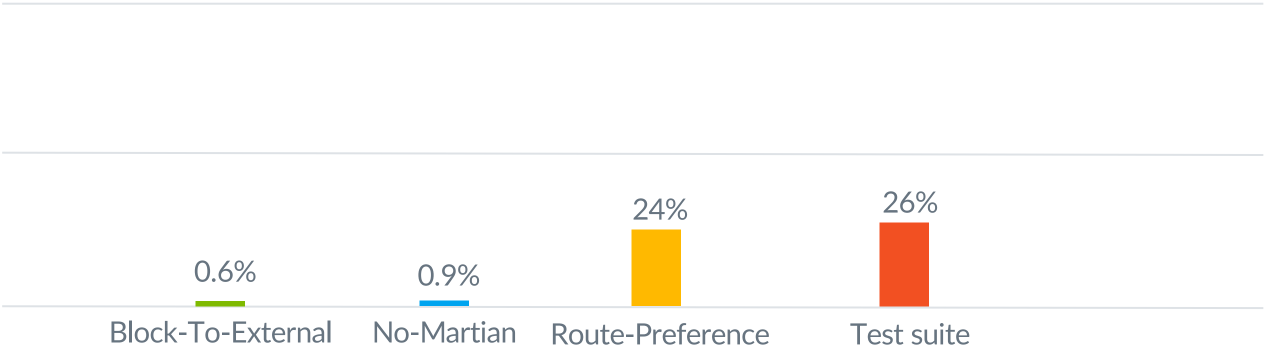
- Bagpipe[3] verified Internet2 BGP configuration with 3 tests:
    - Block-to-external
    - No Martian
    - Route preference

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

Coverage results of existing tests

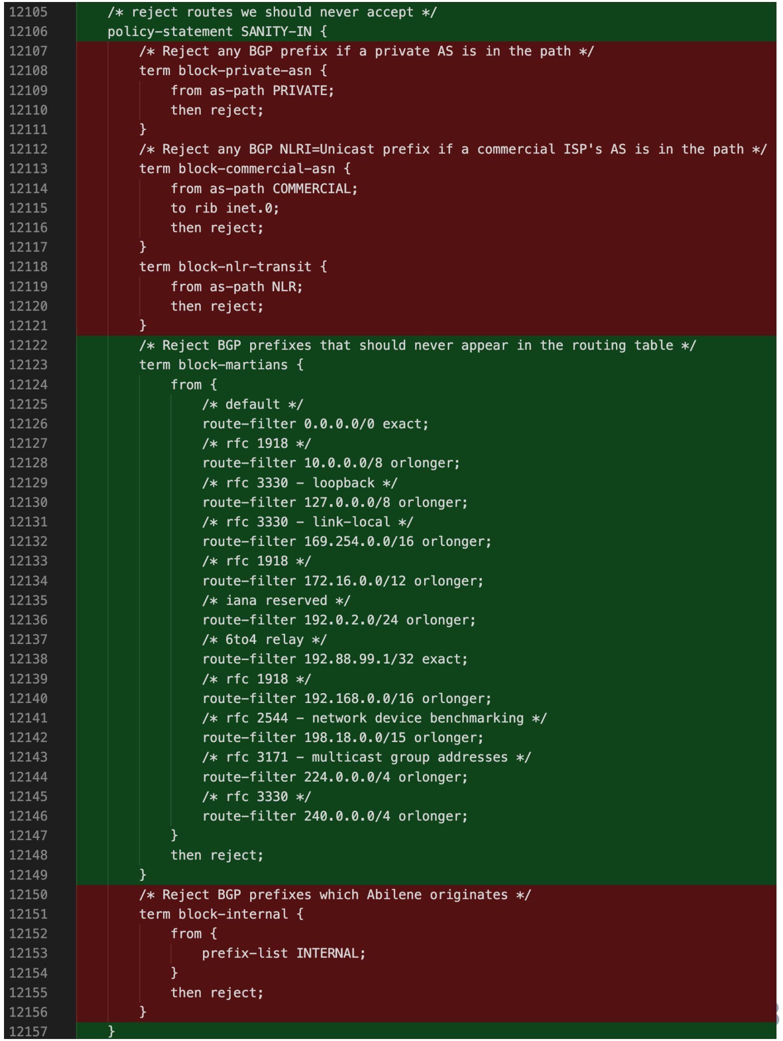
- ▷ The tests left most of the configurations untested.

Fraction of configuration lines covered:



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.

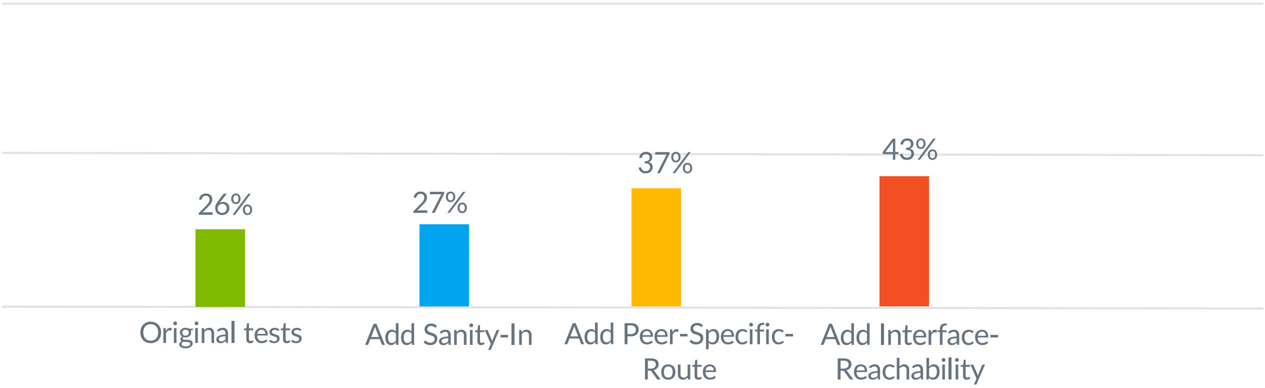


```
/* reject routes we should never accept */
policy-statement SANITY-IN {
    /* Reject any BGP prefix if a private AS is in the path */
    term block-private-asn {
        from as-path PRIVATE;
        then reject;
    }
    /* Reject any BGP NLRI=Unicast prefix if a commercial ISP's AS is in the path */
    term block-commercial-asn {
        from as-path COMMERCIAL;
        to rib inet.0;
        then reject;
    }
    term block-nlr-transit {
        from as-path NLR;
        then reject;
    }
    /* Reject BGP prefixes that should never appear in the routing table */
    term block-martians {
        from {
            /* default */
            route-filter 0.0.0.0/0 exact;
            /* rfc 1918 */
            route-filter 10.0.0.0/8 orlonger;
            /* rfc 3330 - loopback */
            route-filter 127.0.0.0/8 orlonger;
            /* rfc 3330 - link-local */
            route-filter 169.254.0.0/16 orlonger;
            /* rfc 1918 */
            route-filter 172.16.0.0/12 orlonger;
            /* iana reserved */
            route-filter 192.0.2.0/24 orlonger;
            /* 6to4 relay */
            route-filter 192.88.99.1/32 exact;
            /* rfc 1918 */
            route-filter 192.168.0.0/16 orlonger;
            /* rfc 2544 - network device benchmarking */
            route-filter 198.18.0.0/15 orlonger;
            /* rfc 3171 - multicast group addresses */
            route-filter 224.0.0.0/4 orlonger;
            /* rfc 3330 */
            route-filter 240.0.0.0/4 orlonger;
        }
        then reject;
    }
    /* Reject BGP prefixes which Abilene originates */
    term block-internal {
        from {
            prefix-list INTERNAL;
        }
        then reject;
    }
}
```

Coverage was improved effectively

- From 26% to 43% after 3 iterations

Fraction of configuration lines covered:



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
```