The present and future of network verification

Ratul Mahajan

UW CSE 561, Winter 2021

June 15, 2020 T-Mobile
Network Outage Report

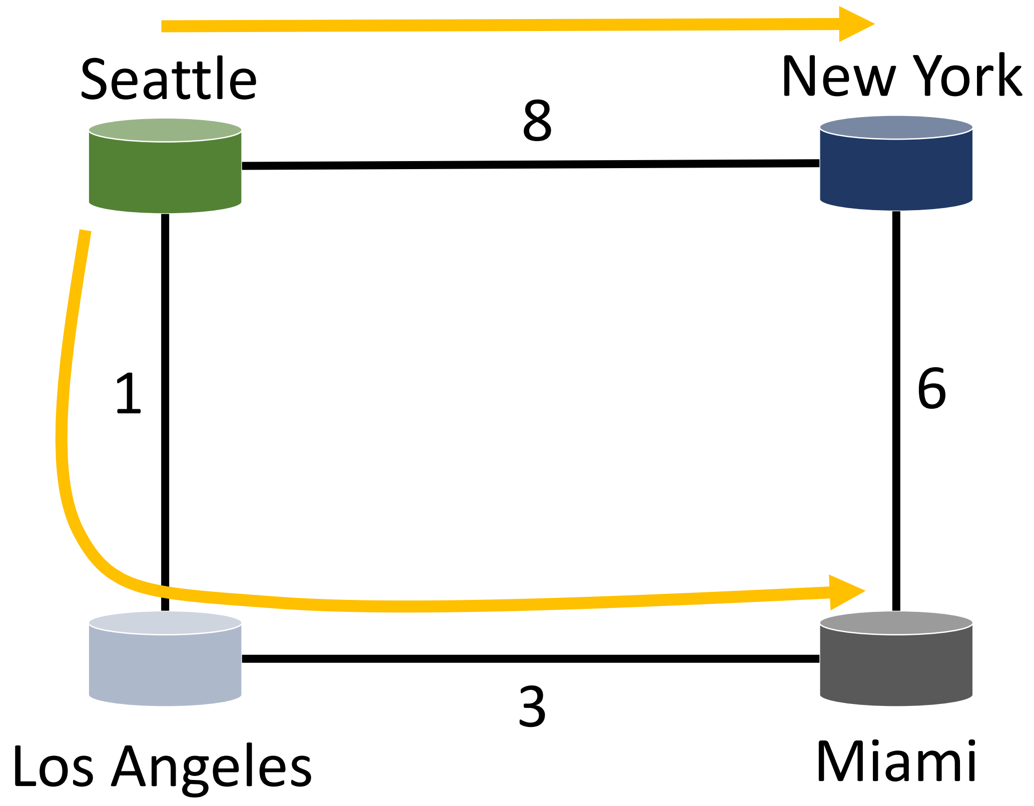
PS Docket No. 20-183

A Report of the Public Safety and Homeland Security Bureau
Federal Communications Commission
October 22, 2020

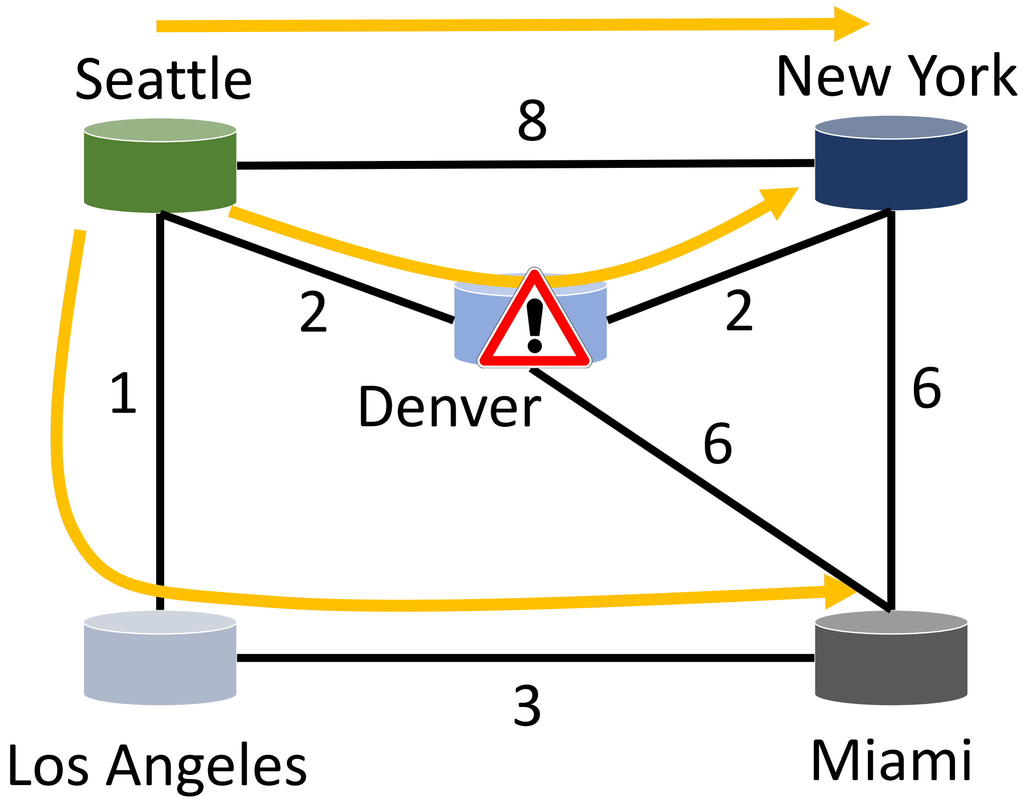
“At least 41% of all calls that attempted to use T-Mobile’s network during the outage failed, including at least 23,621 failed calls to 911.”

“[An old woman] who has dementia, could not reach [her son] after her car would not start and her roadside-assistance provider could not call her to clarify her location; she was stranded for seven hours”

Anatomy of the outage (illustration)

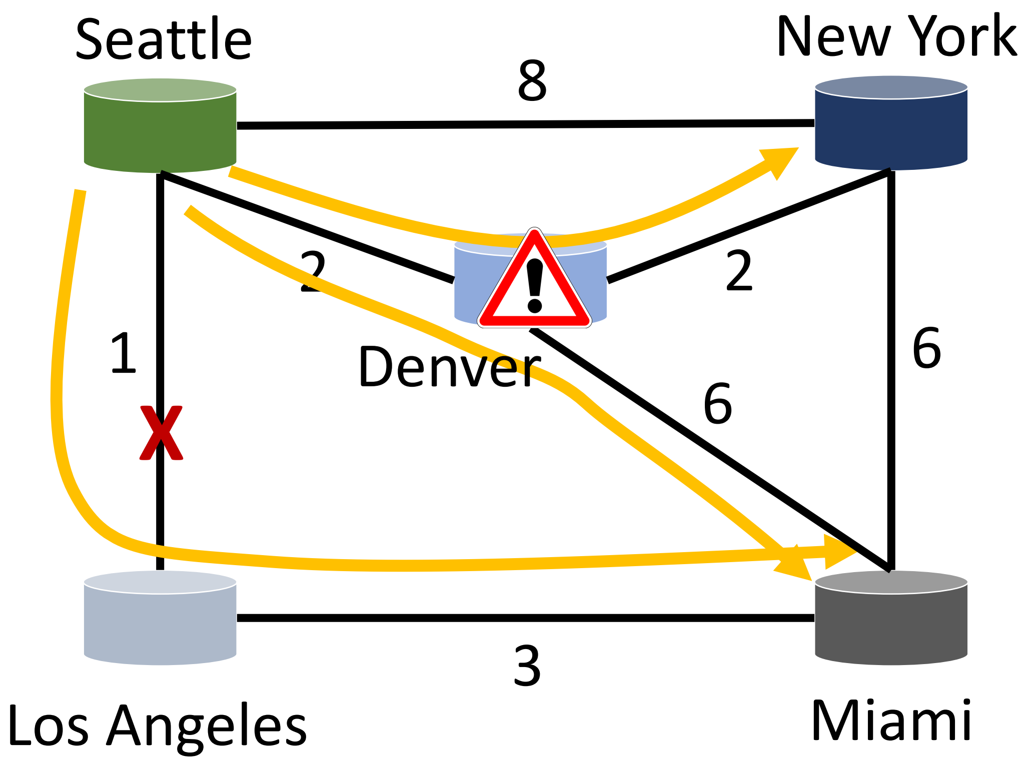


Anatomy of the outage (illustration)



Anatomy of the outage (illustration)

What if T-Mobile could guarantee that no traffic will transit Denver?



What if T-Mobile could predict the impact of link failure?

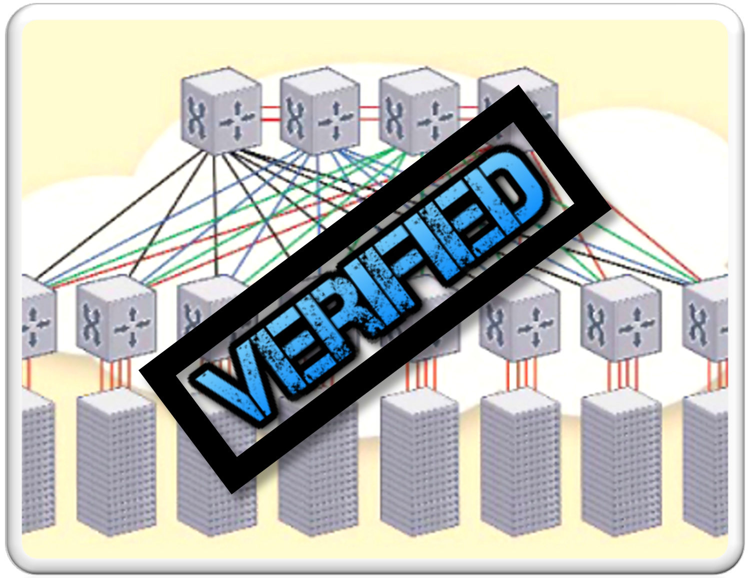
Network verification

Guarantee network behavior*†

* *Some aspect of behavior*

† *Under some assumptions*

A horizontal slice of the problem



Configuration, state — Verify

Software (OS, protocols) — Trust

Hardware

The "haystack" of network behaviors is HUGE

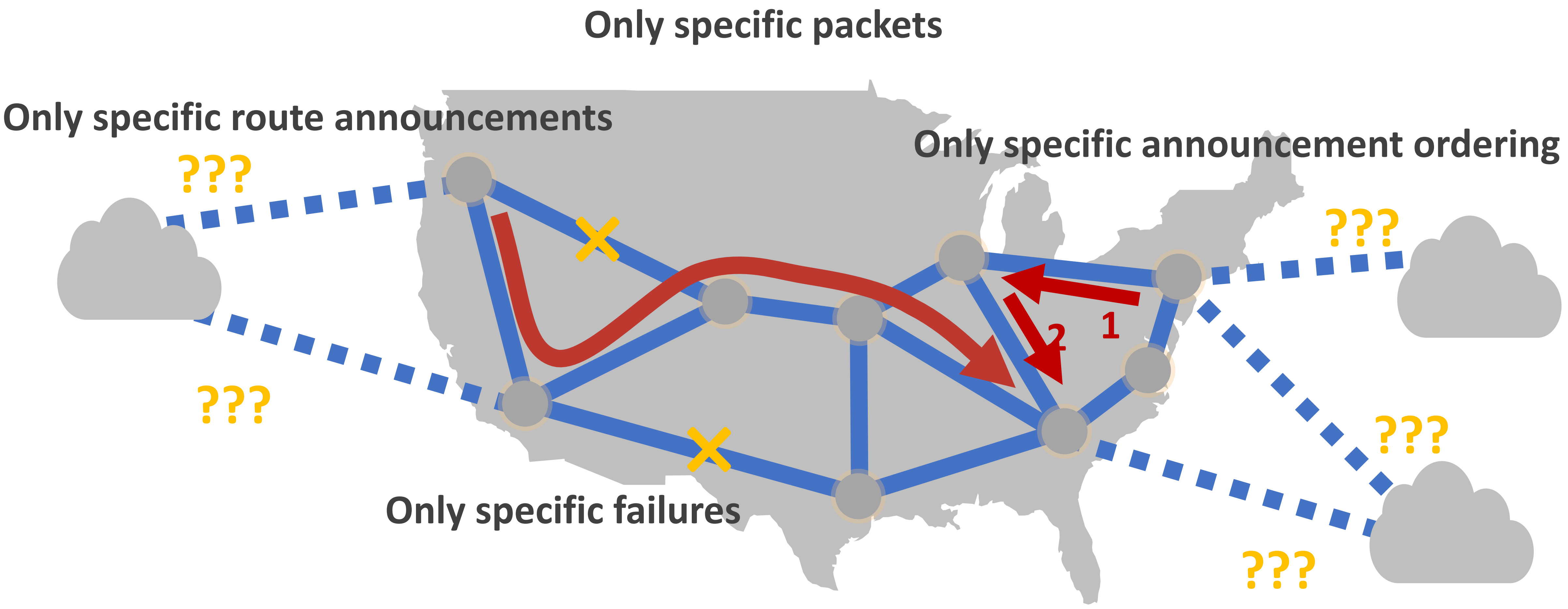
Large scale

$O(10^3)$ devices
$O(10^4)$ config lines / device
$O(10^6)$ FIB entries / device

Complex interactions

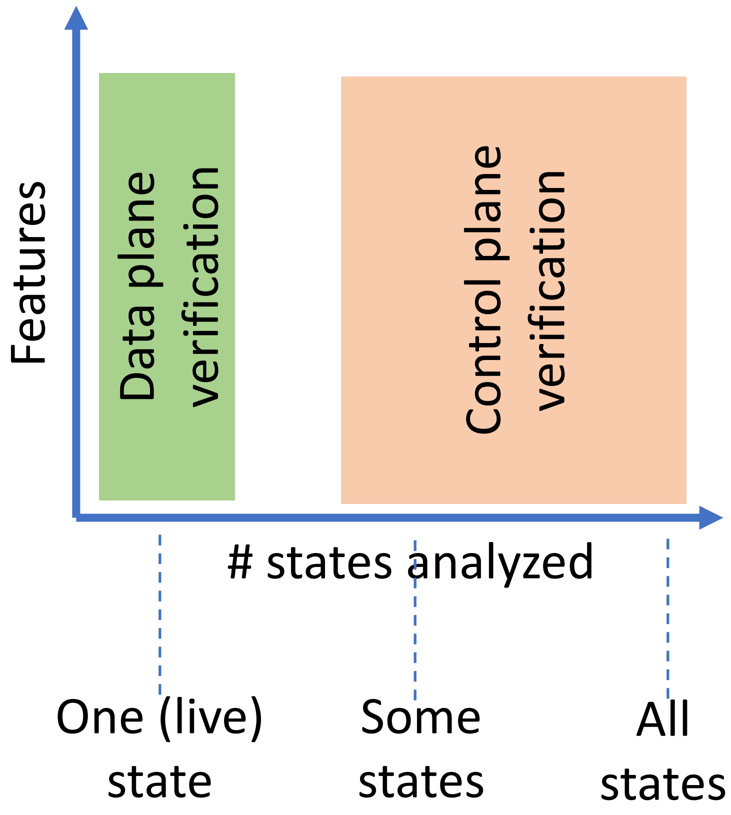
Distributed routing
Protocol redistribution
Rich route filters

The "needles" in network behavior are tiny



The 2D space of network verification tools

Shortest-path or policy routing?
Are packet transformed?
Stateless or stateful forwarding?
…

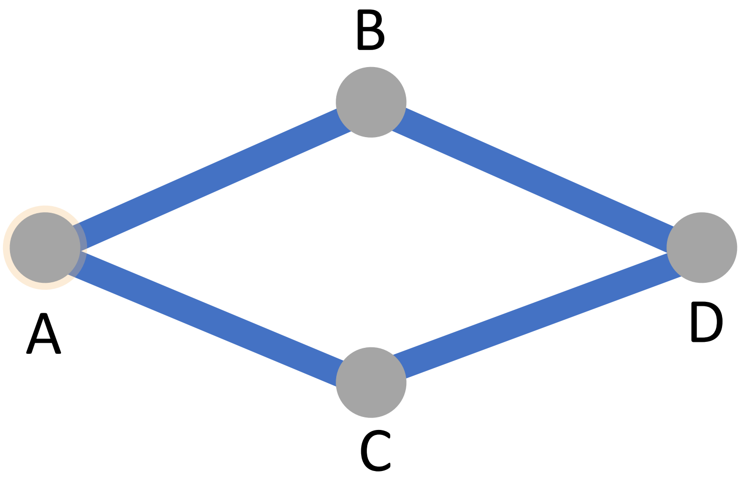


Data plane verification

Who can talk to whom using which ***packets*** and **paths** in **one state** of the network?

10.10.2.0/24	A
0.0.0.0/0	D

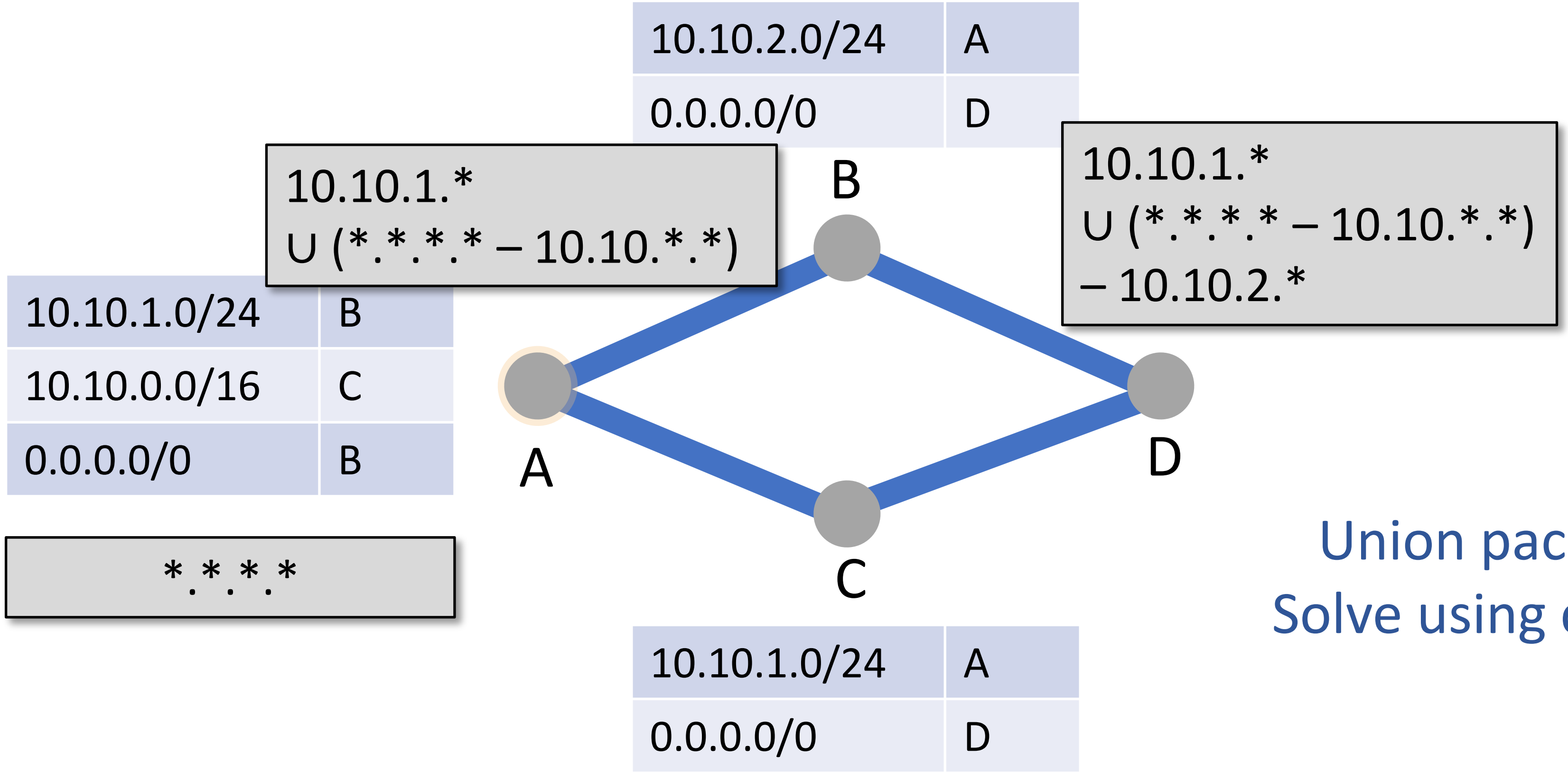
10.10.1.0/24	B
10.10.0.0/16	C
0.0.0.0/0	B



10.10.1.0/24	A
0.0.0.0/0	D

Can A talk to D and using which packets?

DPV idea: Ternary simulation



Union packet sets along possible paths

Solve using custom data structure or BDDs

HSA [2012]

More DPV

Alternative methods: Xie et al. [2005], Anteater [2011], Atomic predicates [2013]

Scalability in specific settings: Parallelism [Libra 2014], Symmetry [2016], local checks [RCDC 2019]

Incrementality: NetPlumber [2013], VeriFlow [2013], Delta-net[2017]

Stateful processing: VMN [2017], SymNet [2016], NetSMC [2020]

Programmable data planes: p4v [2018]

Stateless DPV is a “solved problem”

Stateful and programmable DPV not there yet

Control plane verification

Who can talk to whom using which ***packets*** and **paths** in **many states** of the network?

Finds bugs proactively
Enables what if analysis

Verifying distributed control planes

Routers generate and process messages per low-level directives

```
ospf interface int2_1 metric 1
ospf interface int2_1 metric 1
ospf redistributed connected metric 10
```

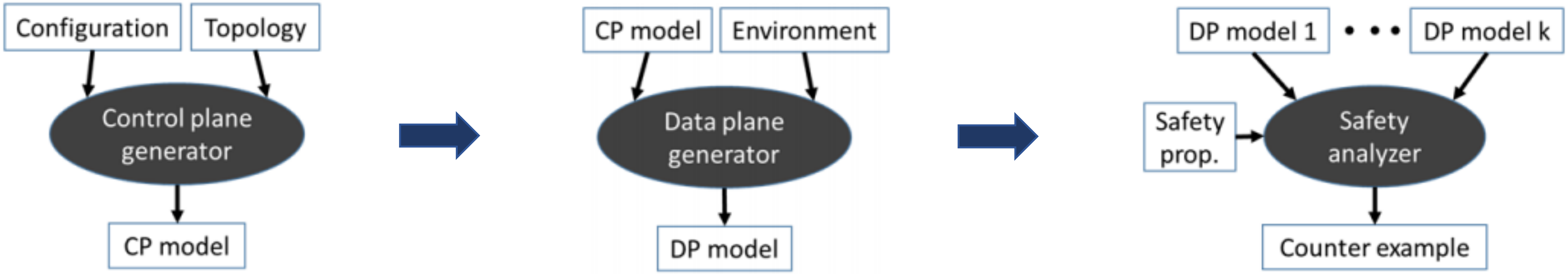
```
ip prefix-list PL1 deny 192.168.0.0/16 le 32
ip prefix-list PL1 allow
route-map FromR2 10
  match ip address prefix-list PL1
  set local-preference 120
```

Goal

Reason about states that emerge when many such programs run concurrently

CPV idea #1: Simulate the control plane

1. Simulate the control plane to generate data plane states
2. Use DPV to analyze the states



Can analyze *any* data plane but not *all* data planes?

Batfish [2015]

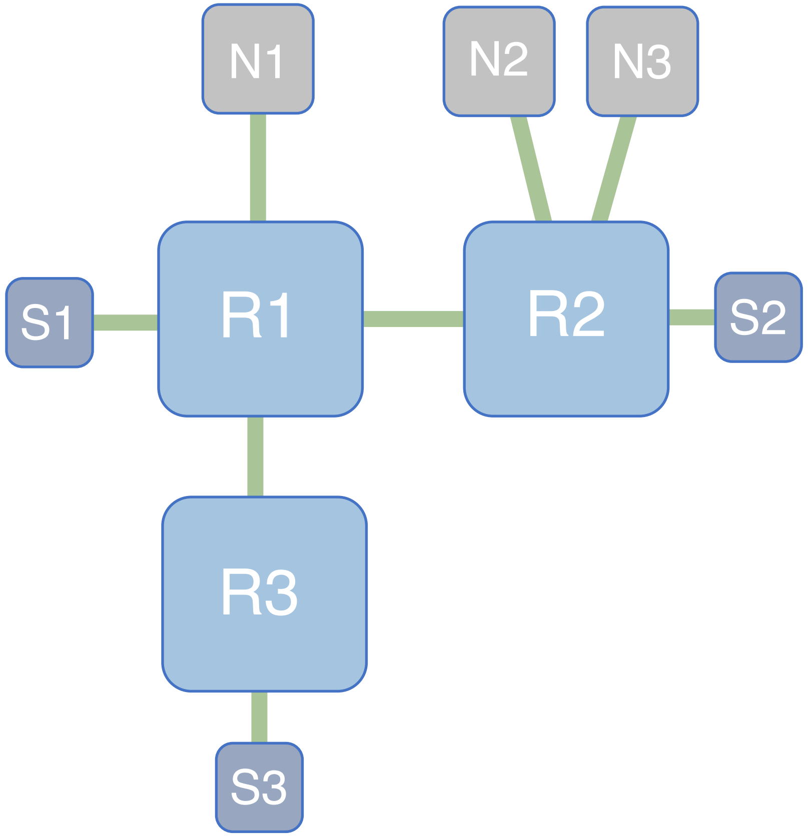
CPV idea #2: Encode the fixed point

1. Valid network states are fixed points of the control plane
2. Fixed points can be formally encoded

ARC [2016] use a graph encoding (not general)

Minesweeper [2017] uses SMT encoding

Minesweeper overview



"Does P hold in the network?"

Network encoding: N

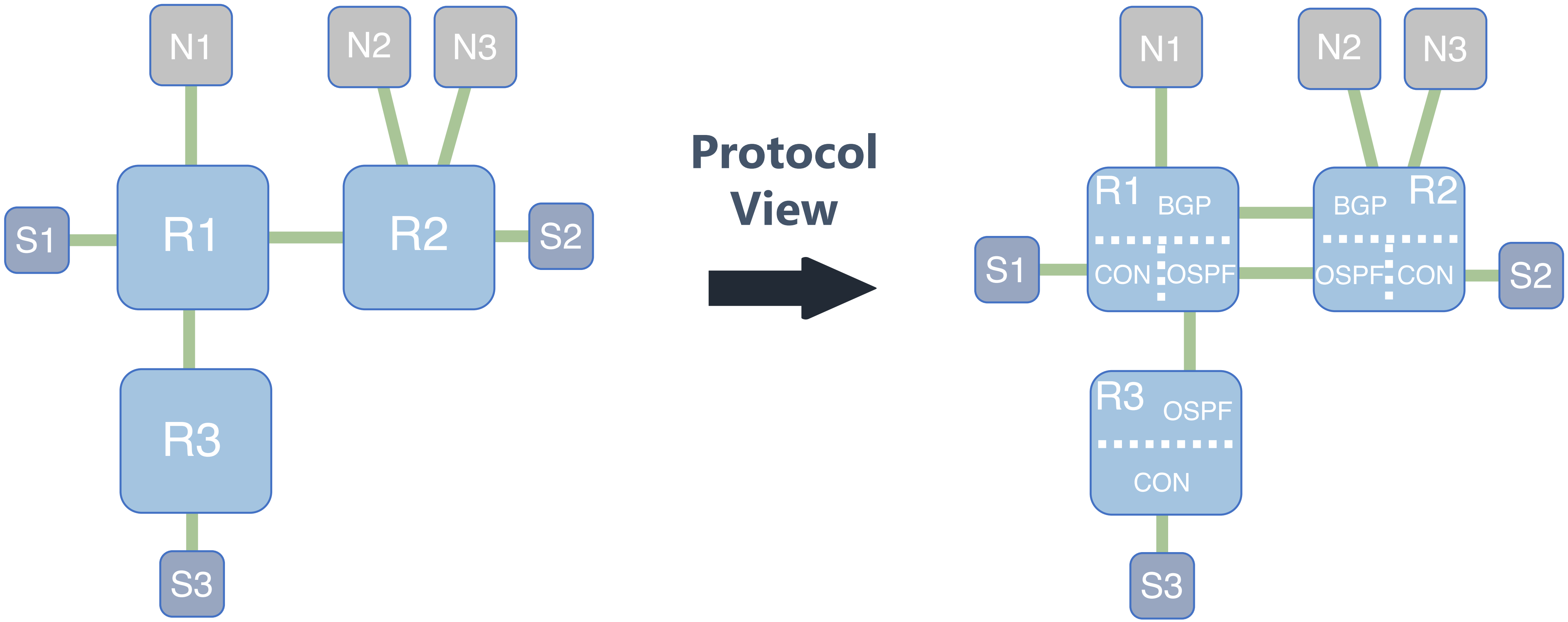
$\wedge$

Property: $\neg$ P

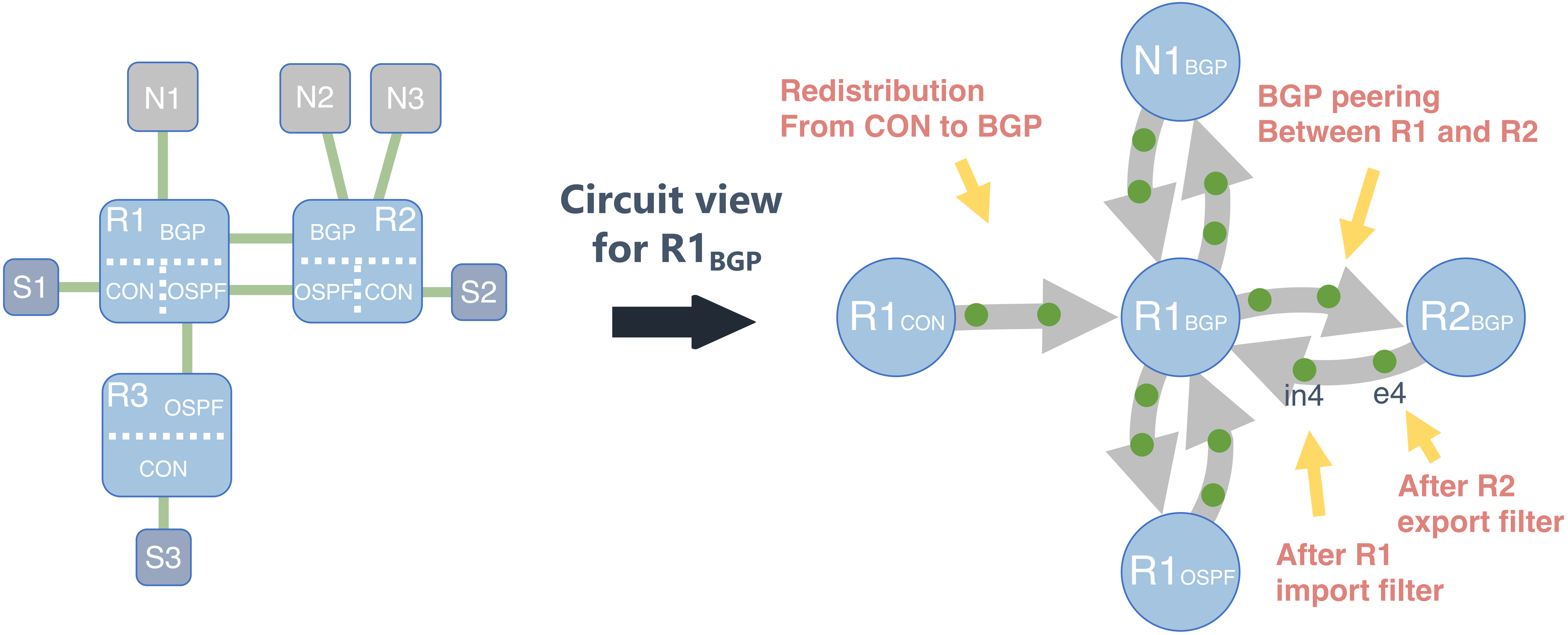
Satisfiable: Property violation

Unsatisfiable: Property holds for all states (or the network does not converge)

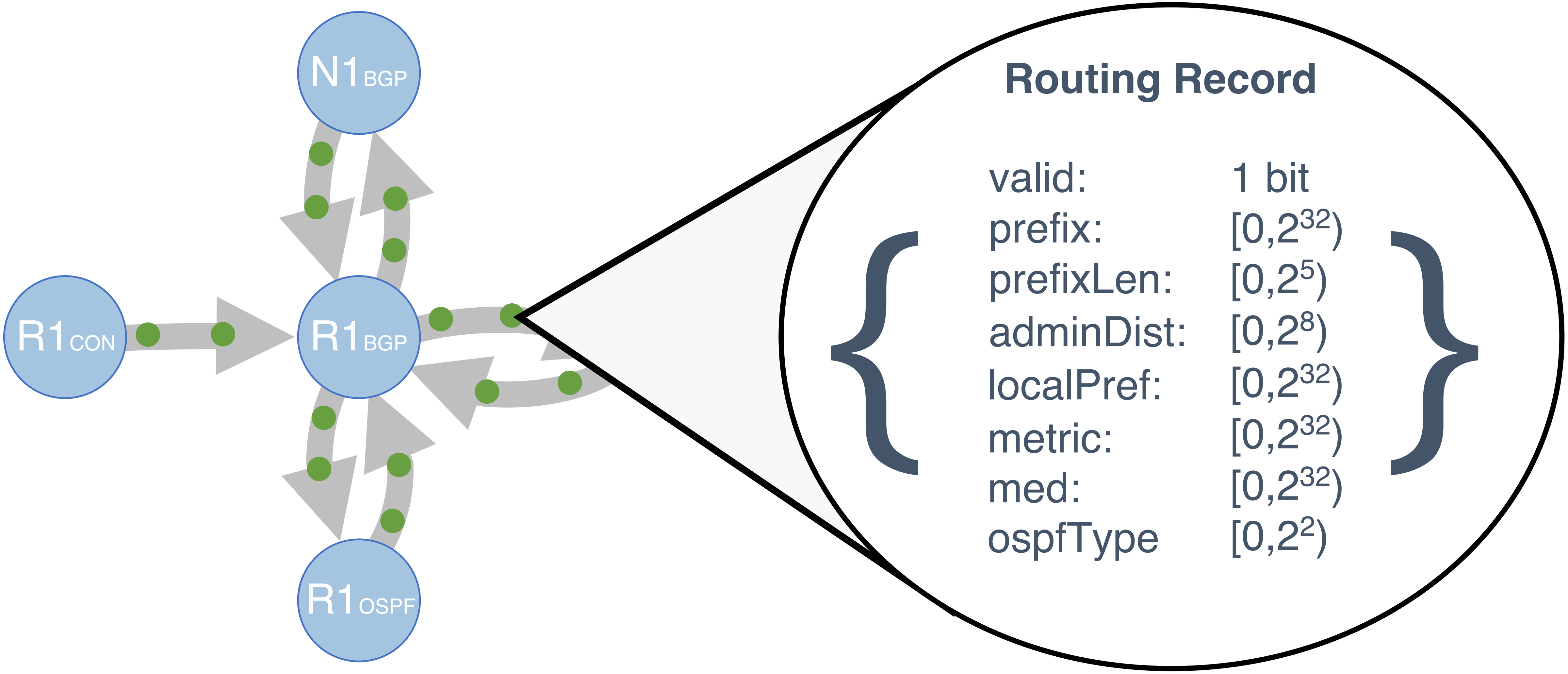
1 Encode protocol interactions



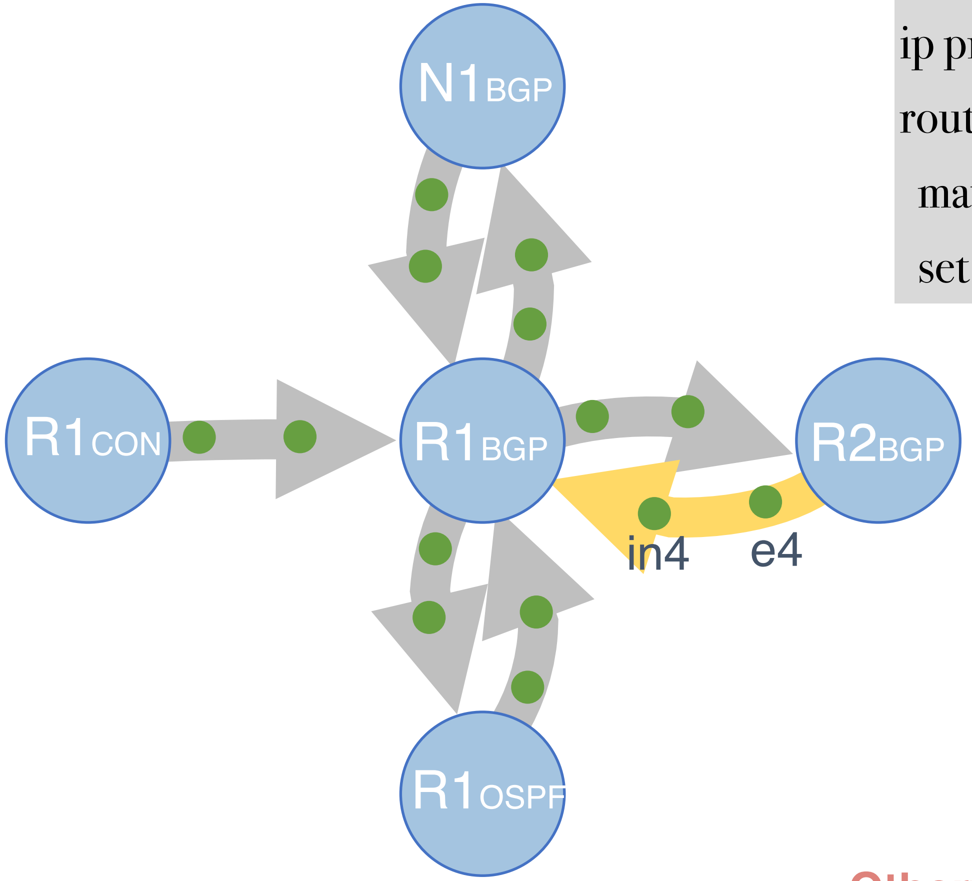
1 Encode protocol interactions



2 Encode routing messages



2 Encode routing messages



Import filter on R1 from R2

```
ip prefix-list PL1 deny 192.168.0.0/16 le 32
ip prefix-list PL1 allow
route-map FromR2 10
 match ip address prefix-list PL1
 set local-preference 120
```

If R2 exports a route

And it passes the import filter

if $e_4.\text{valid} \wedge \text{failed}_{R1,R2} = 0 \wedge$
$\neg(\text{FBM}(e_4.\text{prefix}, 192.168.0.0, 16) \wedge$
$16 \leq e_4.\text{prefixLen} \leq 32)$ **then**

Then R1 has the same route with local preference of 120

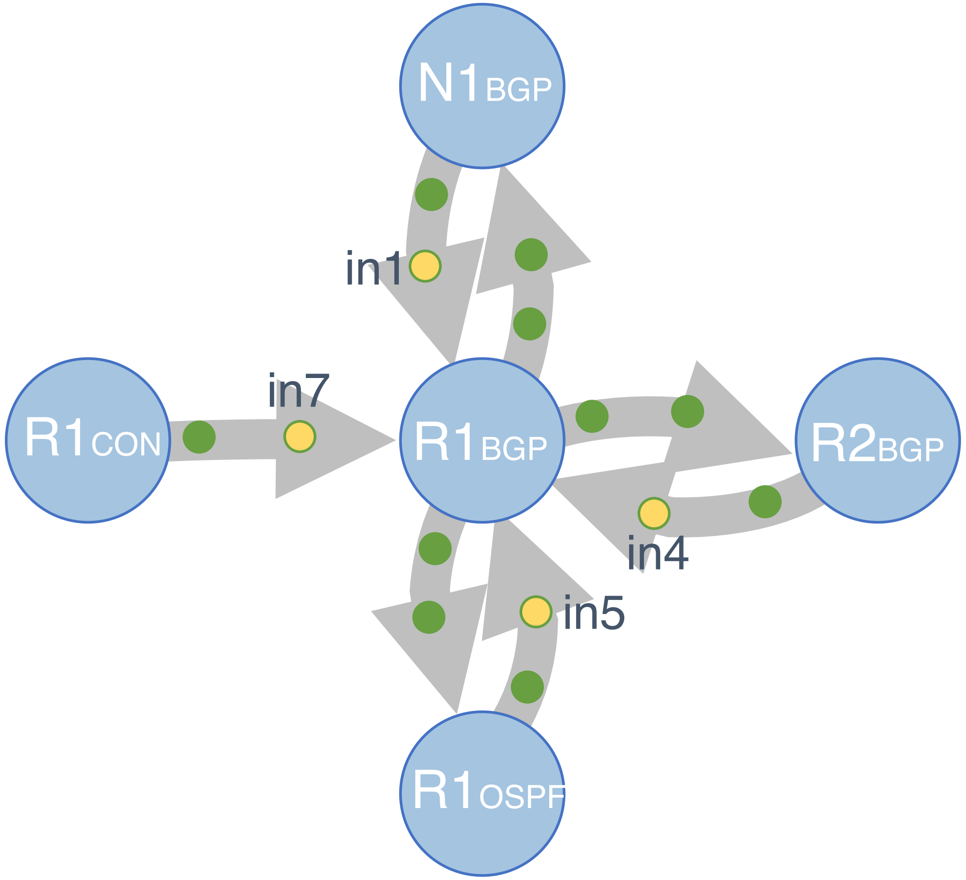
$in_4.\text{valid} = \text{true}$
$in_4.\text{lp} = 120$
$in_4.\text{prefix} = e_4.\text{prefix}$
$in_4.\text{metric} = e_4.\text{metric}$
$in_4.\text{prefixLen} = e_4.\text{prefixLen}$

…

Otherwise, R1 has no route from R2

else $in_4.\text{valid} = \text{false}$

3 Encode routing decisions



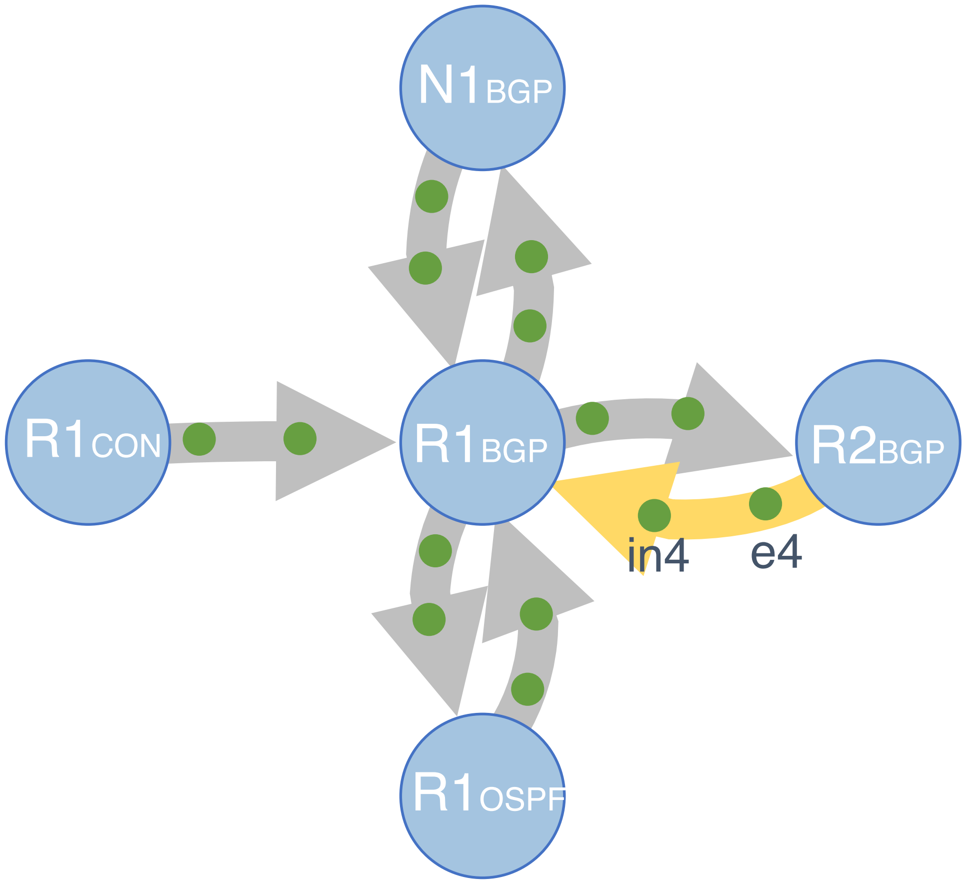
What is the best BGP route?

$$R1_{BGP}\text{-BEST} = \text{Min}(in1, in4, in5, in7)$$

What is the best overall route?

$$R1\text{-BEST} = \text{Min}(R1_{BGP}\text{-BEST}, R1_{OSPF}\text{-BEST}, R1_{CON}\text{-BEST})$$

3 Encode routing decisions



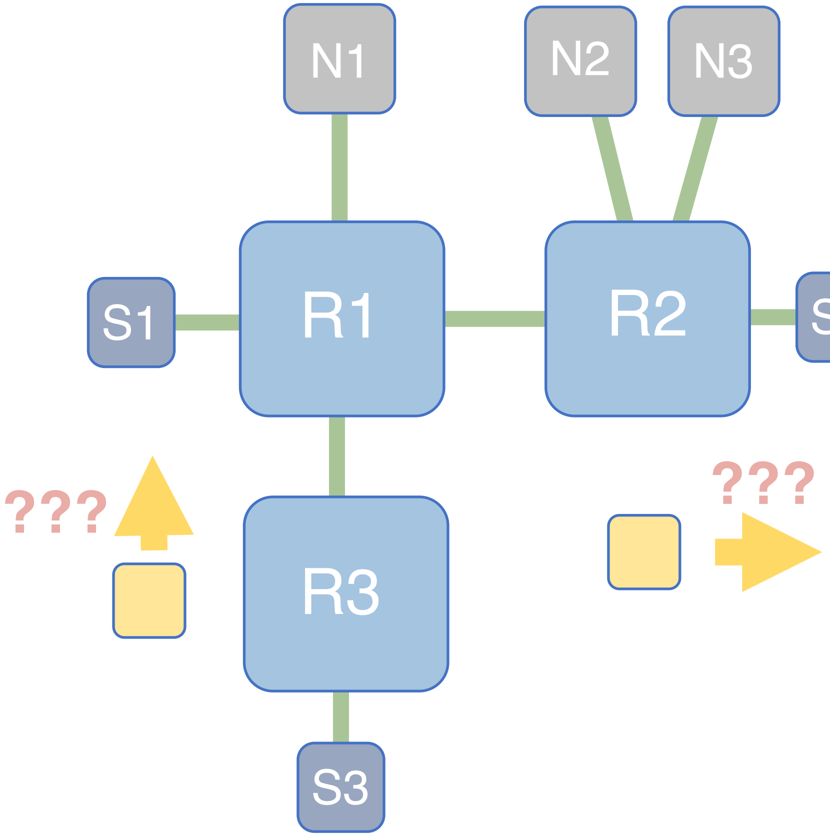
Does R1 have a RIB entry for R2?

$$RIB_{R1,R2} \;=\; (in4 = R1\text{-}Best)$$

Does R1 have a FIB entry for R2?

$$FIB_{R1,R2} \;=\; RIB_{R1,R2} \wedge \neg\, ACL(R1,R2)$$

4 Encode the data packet



Symbolic Packet

$$
\left\{
\begin{array}{cccc}
0 & \leq & \text{dstIp} & \leq 2^{32} \\
0 & \leq & \text{srcIp} & \leq 2^{32} \\
0 & \leq & \text{dstPort} & \leq 2^{16} \\
0 & \leq & \text{srcPort} & \leq 2^{16} \\
0 & \leq & \text{protocol} & \leq 2^{8}
\end{array}
\right\}
$$

5 Encode the property

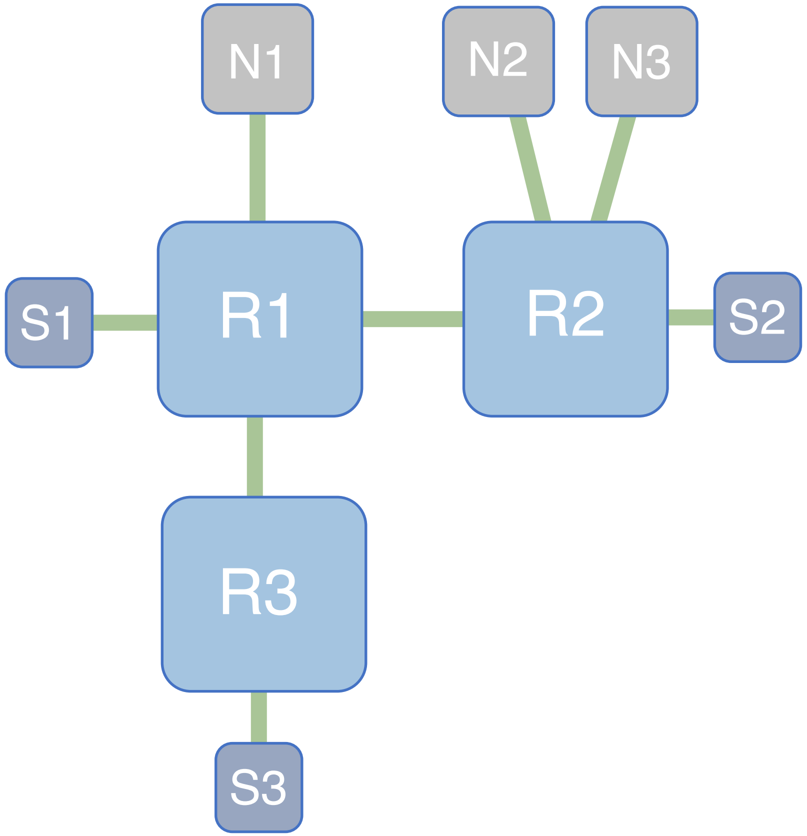
> “Can R3 reach subnet S2?”

$\text{canReach}_{R2} \Leftrightarrow \text{FIB}_{R2,S2}$

$$\text{canReach}_{R1} \Leftrightarrow (\text{FIB}_{R1,R2} \wedge \text{canReach}_{R2}) \vee (\text{FIB}_{R1,R3} \wedge \text{canReach}_{R3})$$

$$\text{canReach}_{R3} \Leftrightarrow (\text{FIB}_{R3,R1} \wedge \text{canReach}_{R1})$$

Property: canReach_{R3}



Optimizing the encoding

Must use domain knowledge

SMT solvers cannot automagically perform even “straightforward” optimizations

Prefix Hoisting

Routing Record

valid:	1 bit
~~prefix:~~	~~$[0,2^{32})$~~
prefixLen:	$[0,2^{5})$
adminDist:	$[0,2^{8})$
localPref:	$[0,2^{32})$
metric:	$[0,2^{32})$
med:	$[0,2^{32})$
ospfType	$[0,2^{2})$

FBM(**r.prefix**, 192.4.0.0, 16) $\wedge$
$16 \leq$ r.prefixLen ≤ 32

$$=$$

FBM(**dstIp**, 192.4.0.0, 16) $\wedge$
$16 \leq$ r.prefixLen ≤ 32

$$=$$

(192.4.0.0 $\leq$ **dstIp** $\leq$ 192.4.0.0 + 2^{32-16}) $\wedge$
$16 \leq$ r.prefixLen ≤ 32

Integer Difference Logic!

Slicing

Symbolic Record

valid:	1 bit
prefixLen:	$[0,2^5)$
~~adminDist:~~	~~$[0,2^8)$~~
~~localPref:~~	~~$[0,2^{32})$~~
metric:	$[0,2^{32})$
~~med:~~	~~$[0,2^{32})$~~
~~ospfType~~	~~$[0,2^2)$~~

Statically analyze configs
to infer irrelevant attributes

Local preference has no
impact if never explicitly set

Does it work for real networks?

152 networks

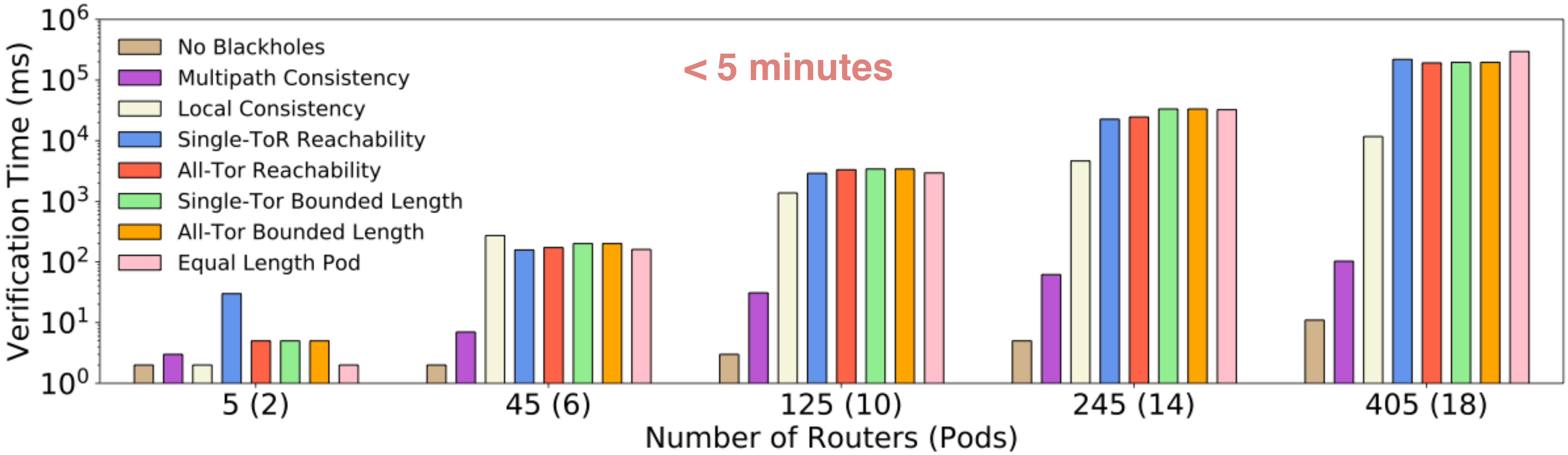
OSPF, eBGP, iBGP
Static routes
ACLs
Route redistribution

120 violations

Hijack-able internal addresses
Asymmetric paired ACLs

Proved fault tolerance

How well does it scale?



More CPV

Tradeoff generality for scalability

- ARC [2016], Tiramisu [2019], FastPlane [2019], Shapeshifter [2020]

Model checking

- Plankton [2019]

Exploit symmetry

- Bonsai [2018], Origami [2019]

Fully general CPV that scales to
thousands of nodes is still an open problem

Reflections on research to practice journey

Remarkably short path from research to practice

All large cloud providers are using it

Startups are enabling broader use

Research to practice gap (1/3)

Network verification is “too complete”

Flags uninteresting violations

Real networks have unwritten assumptions that cause violations

Devices cannot spoof source IPs

Blocking “malicious” IPs or telnet ports is OK

Loosing connectivity under specific failures is OK

Research to practice gap (2/3)

Network verification answers are "too precise"

Hard to explain results to users

Oddly-shaped packet header spaces and network environments

Not easy to map back to user-configured objects

Research to practice gap (3/3)

Network verification is "too different"

Hard for network engineers to use

Testing with concrete inputs is easier than specifying invariants

Inversion of mental model is needed

The future of network verification

Network verification 1.0

Identify promising slices of the problem

Develop core techniques

Network verification 2.0

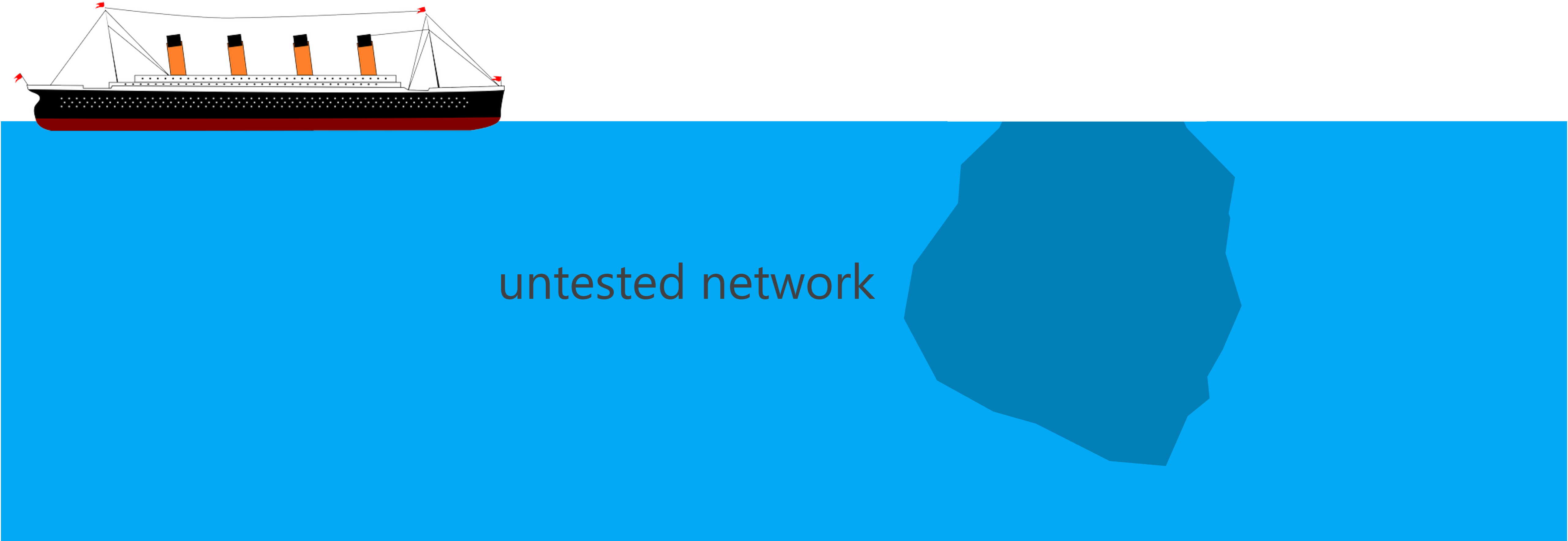
Enable effective use in practice

Enable rapid expansion to more functionality

Verified networks can have outages too

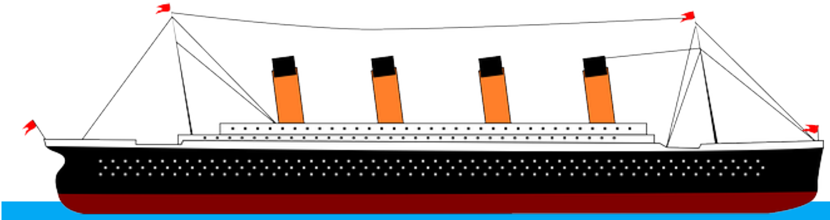
Configuration change

check invariants

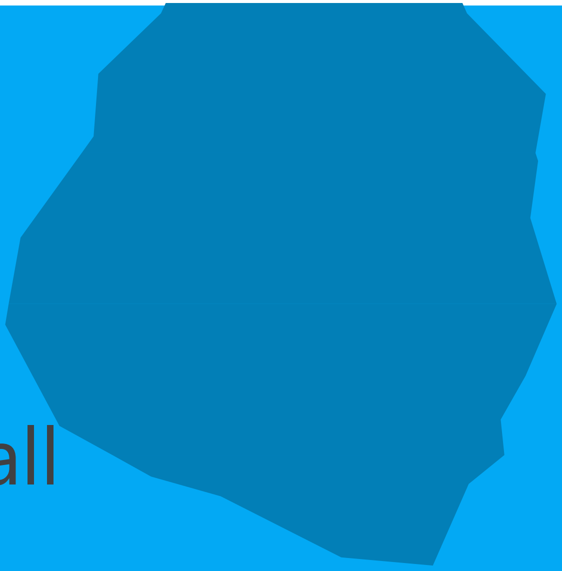


Verified networks can have outages too

Route maintenance
withdraw /18

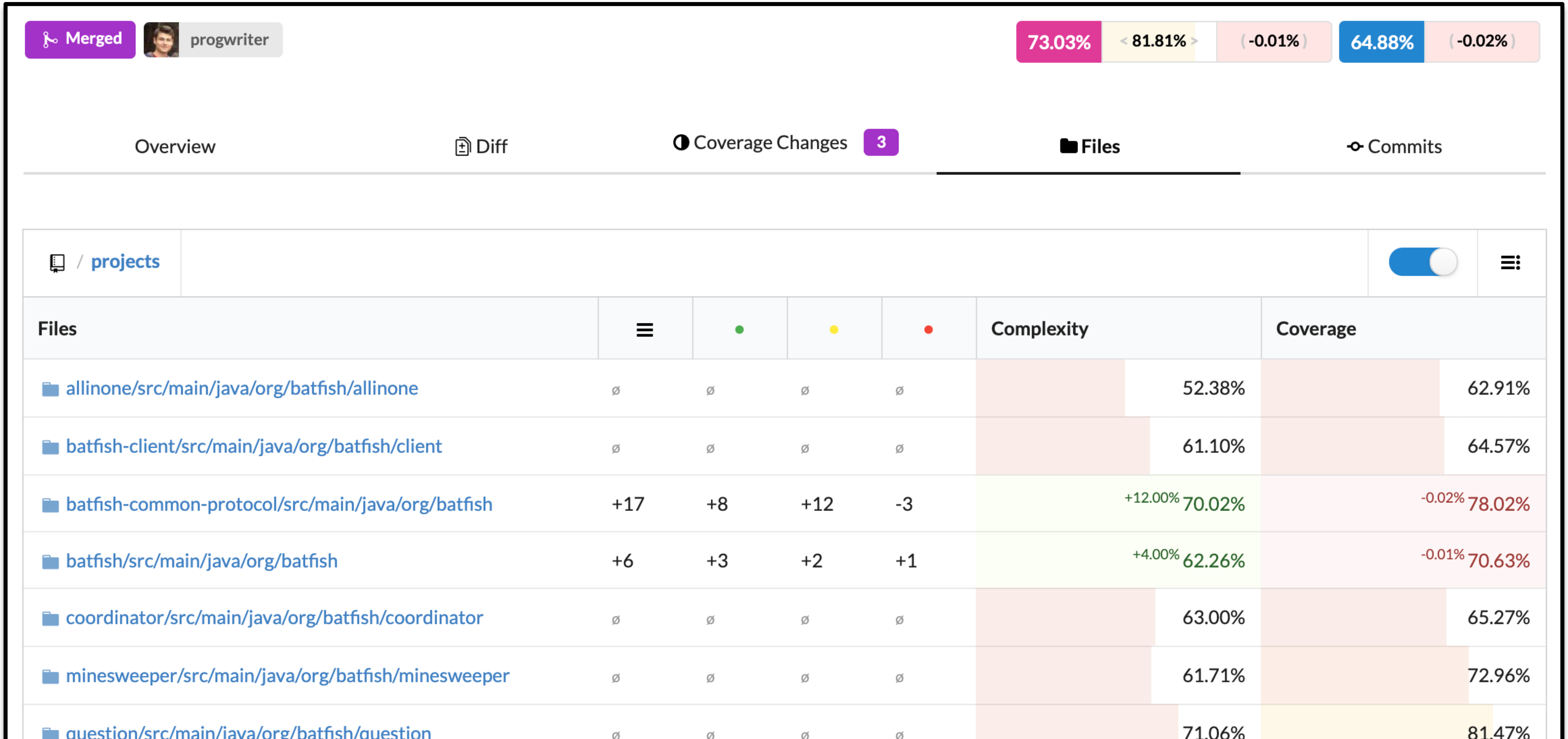


BGP routes
look correct



Traffic carried by
untested static route
and dropped by firewall

Inspiration from software: Code coverage



Files	≡	●	●	●	Complexity	Coverage
allinone/src/main/java/org/batfish/allinone	ø	ø	ø	ø	52.38%	62.91%
batfish-client/src/main/java/org/batfish/client	ø	ø	ø	ø	61.10%	64.57%
batfish-common-protocol/src/main/java/org/batfish	+17	+8	+12	-3	+12.00% 70.02%	-0.02% 78.02%
batfish/src/main/java/org/batfish	+6	+3	+2	+1	+4.00% 62.26%	-0.01% 70.63%
coordinator/src/main/java/org/batfish/coordinator	ø	ø	ø	ø	63.00%	65.27%
minesweeper/src/main/java/org/batfish/minesweeper	ø	ø	ø	ø	61.71%	72.96%
question/src/main/java/org/batfish/question	ø	ø	ø	ø	71.06%	81.47%

How to define “network coverage”?

Unlike programs, network is not a sequence of statements

Model the network as a dependency graph of “facts”

Coverage is % of facts covered (in)directly by defined invariants

Beckett and Mahajan [2019]

Prototype deployed at Microsoft

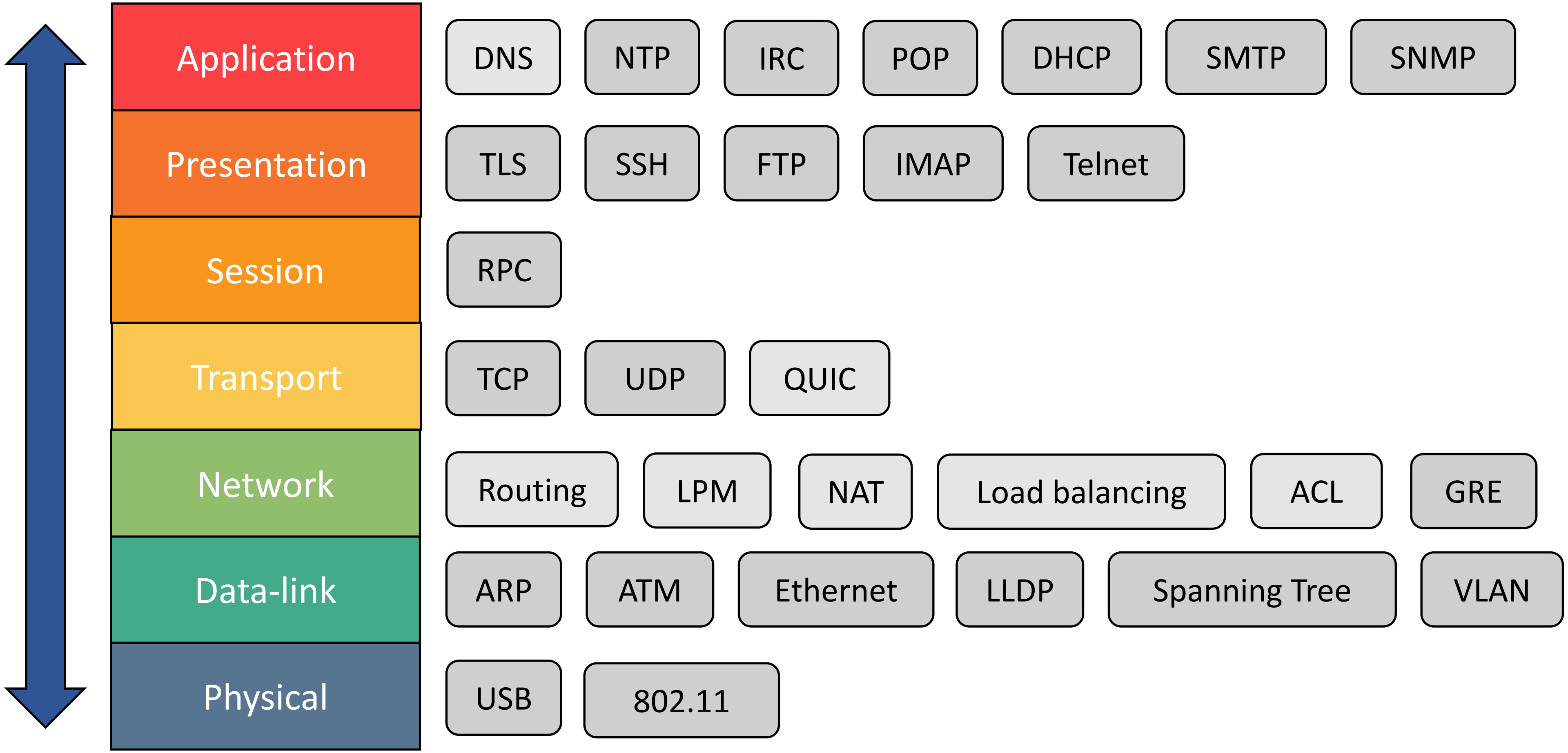
<element #0>	
MethodName	T0Reachability
CoverageSummary	
DeviceLevelCoverage	100.00% (14/14)
FibRuleLevelCoverage	39.39% (52/132)
FibRuleLevelMatchFieldCoverage	39.39% (52/132)
FibRuleLevelActionFieldCoverage	39.39% (52/132)
InterfaceLevelCoverage	80.00% (64/80)
PathCoverage	73.58% (39/53)

The future of network verification

Effective use in practice

Rapid expansion to more functionality

Network stack is broad and deep

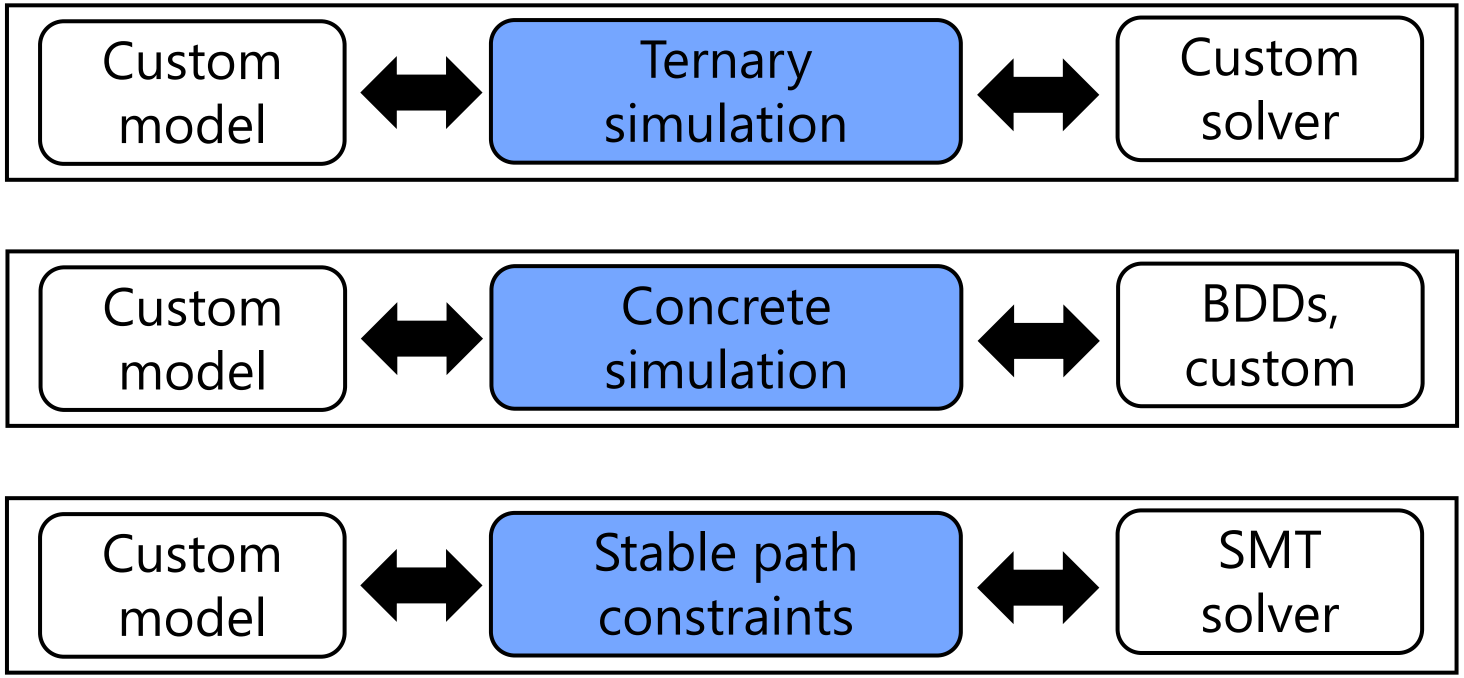


Current monolithic approach does not scale

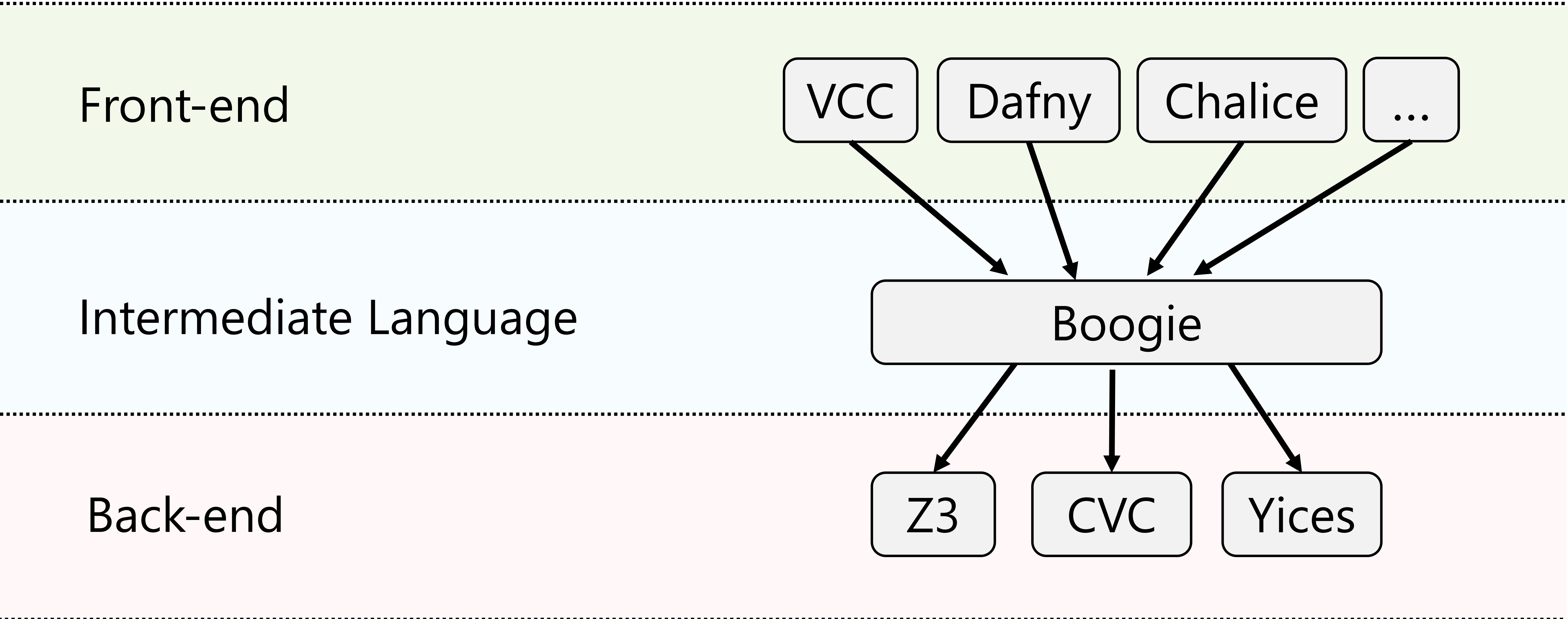
Header Space Analysis
Stateless forwarding

Batfish
Distributed routing

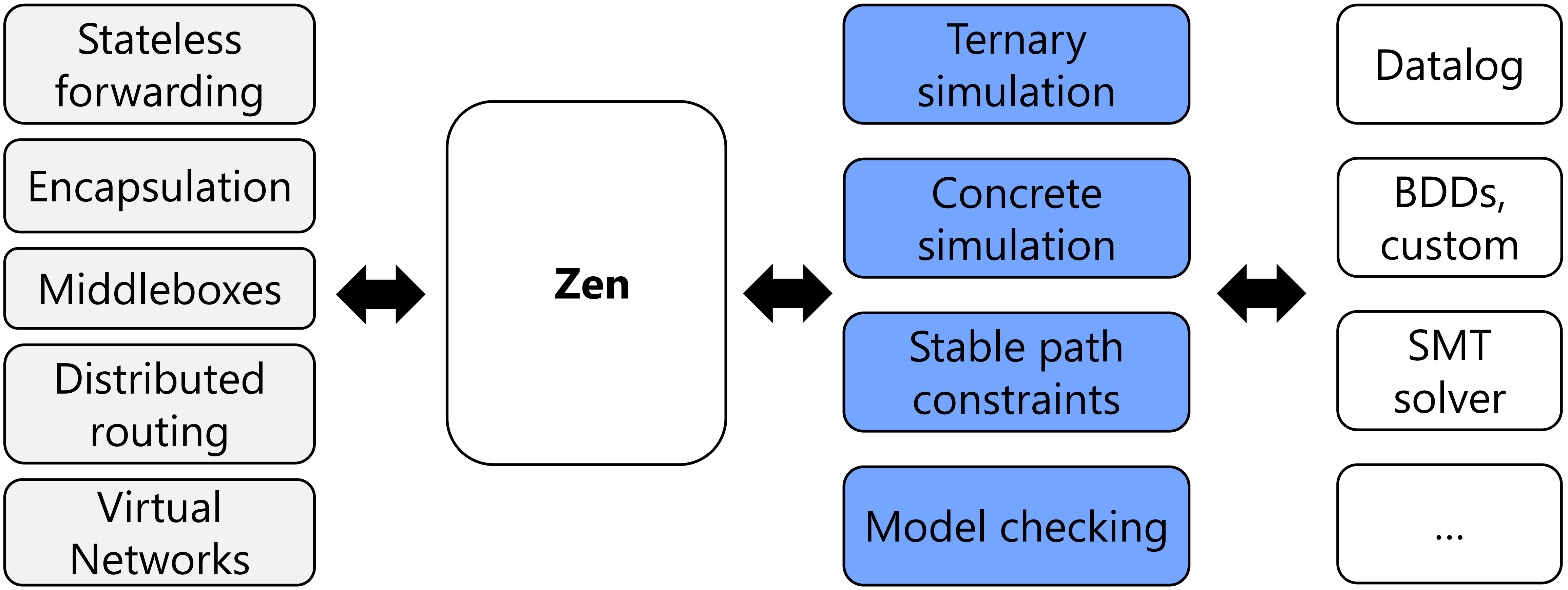
Minesweeper
Distributed routing



Inspiration from software: Intermediate languages



Zen: An IL for network verification



Zen [2020]

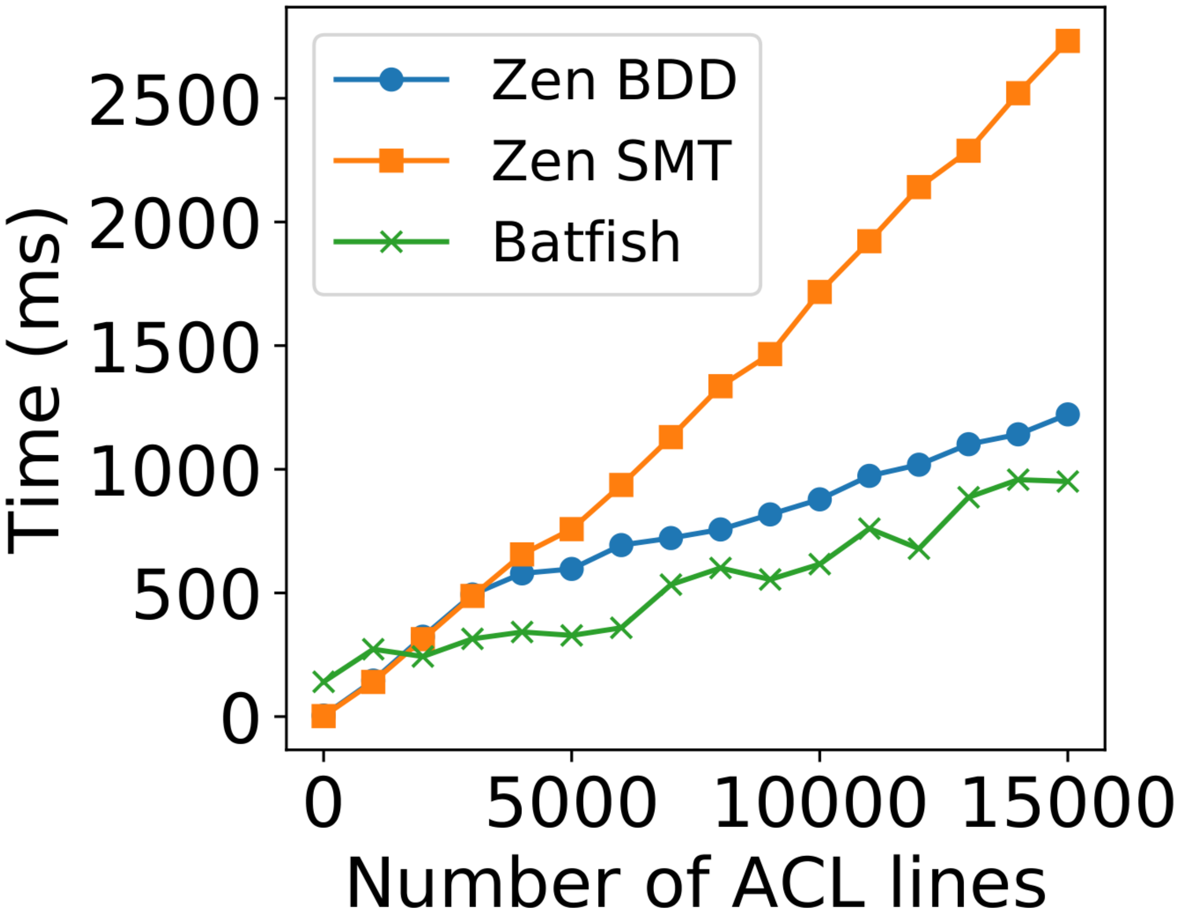
https://github.com/microsoft/zen

Modeling an Azure security group in Zen

Library-provided type `Zen<T>` represents a value that can be symbolic

```
Zen<bool> Allowed(Nsg nsg, Zen<Packet> pkt, int i) {
    if (i >= nsg.Rules.Length)
        return false;
    var rule = nsg.Rules[i];
    return If(Matches(pkt, rule), rule.Permit, Allowed(nsg, pkt, i+1));
}
```

Zen has competitive performance



Summary

Network verification is needed to guarantee correctness for increasingly complex networks

Last few years have seen rapid advances in technology and industry adoption of network verification

The next frontier is enabling effective use and rapidly covering more network functionality