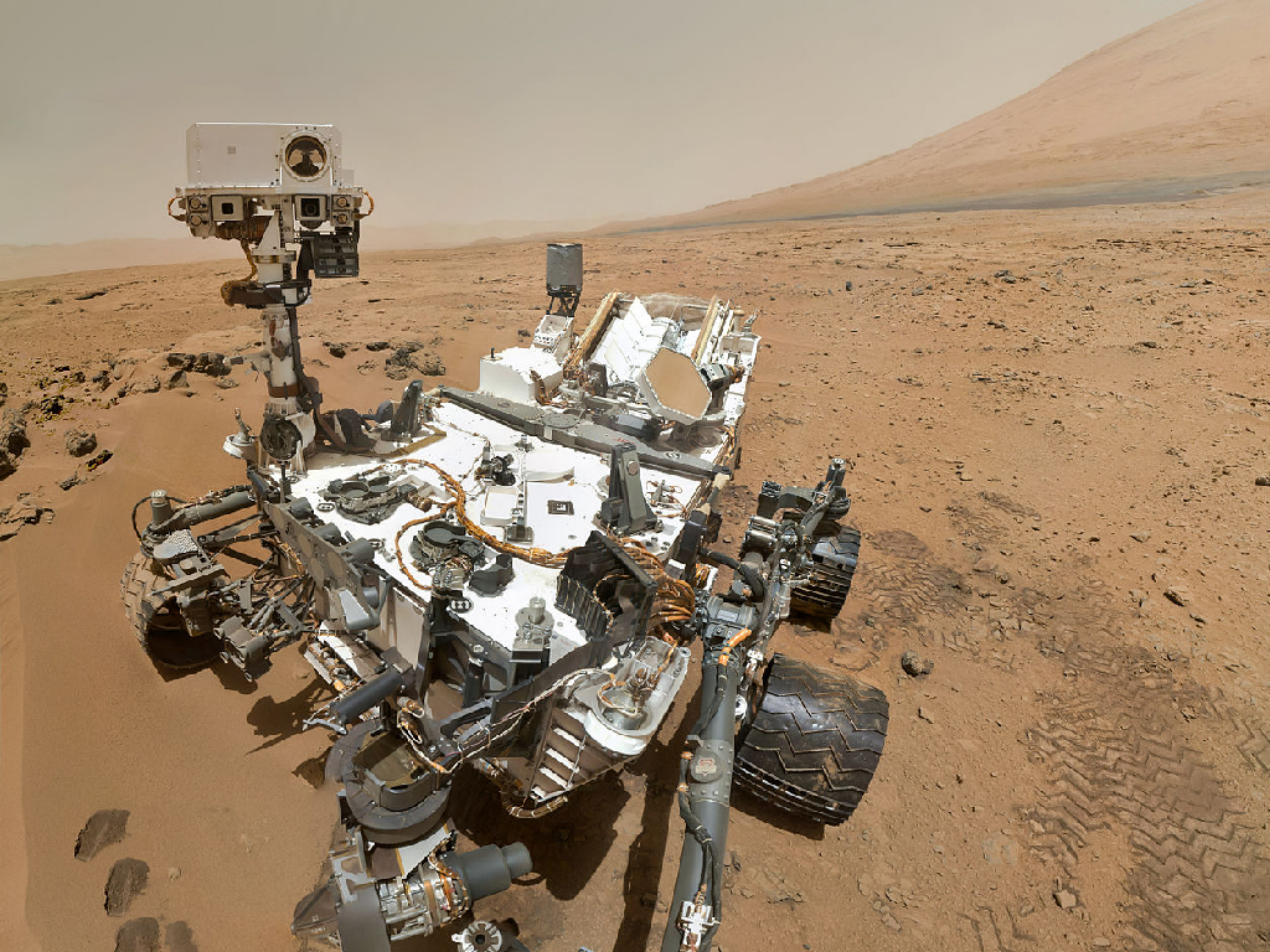
Network Verification: From Algorithms to Deployment

Brighten Godfrey
Associate Professor, UIUC
Co-founder and CTO, Veriflow

2nd Hebrew University Networking Summer
June 21, 2017

Networks are so complex it's hard to know they're doing the right thing.

Let's automate.



Outline for Today

Networking background

Data-plane verification

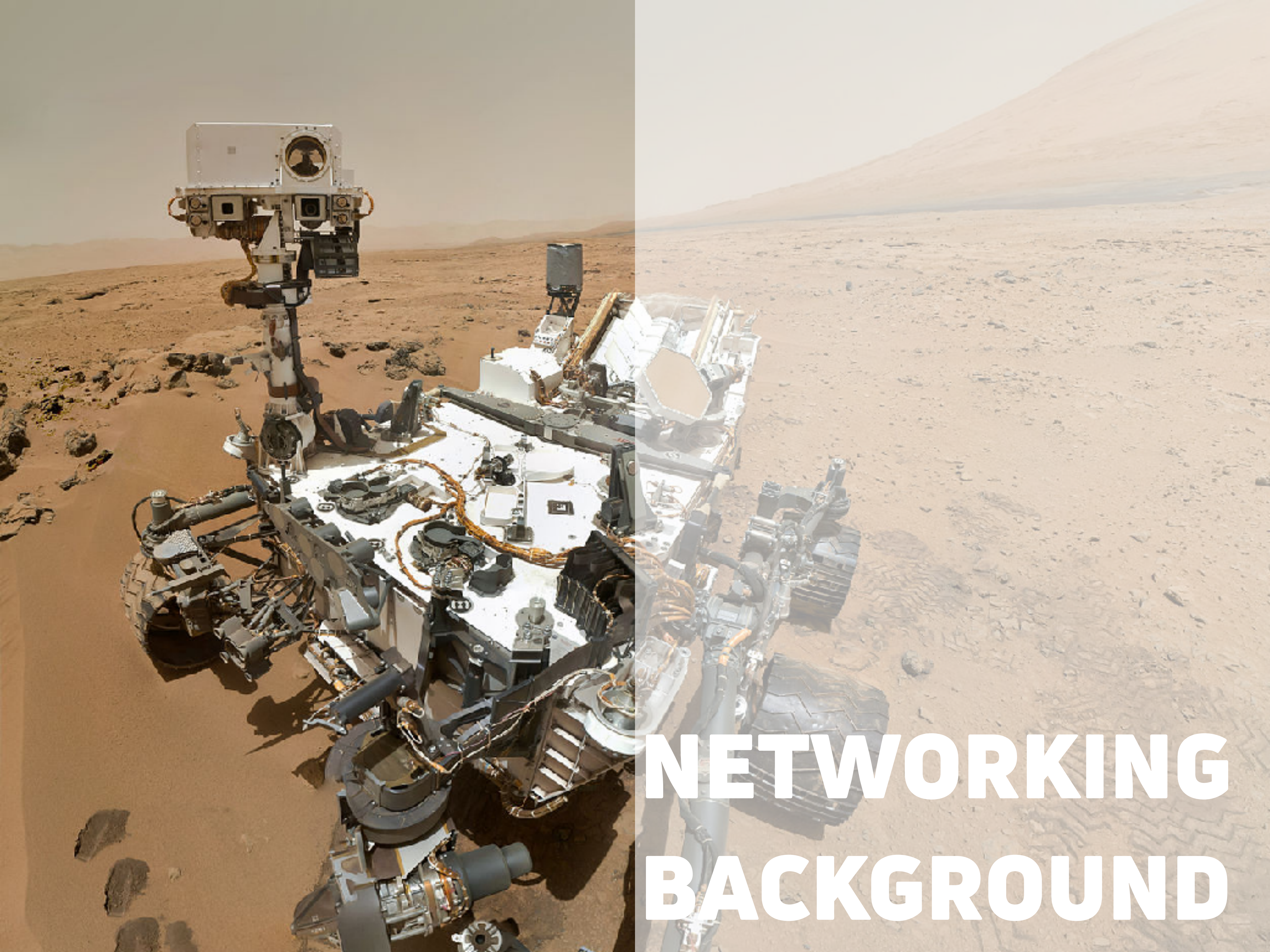
- One-shot
- Real-time incremental

Configuration verification

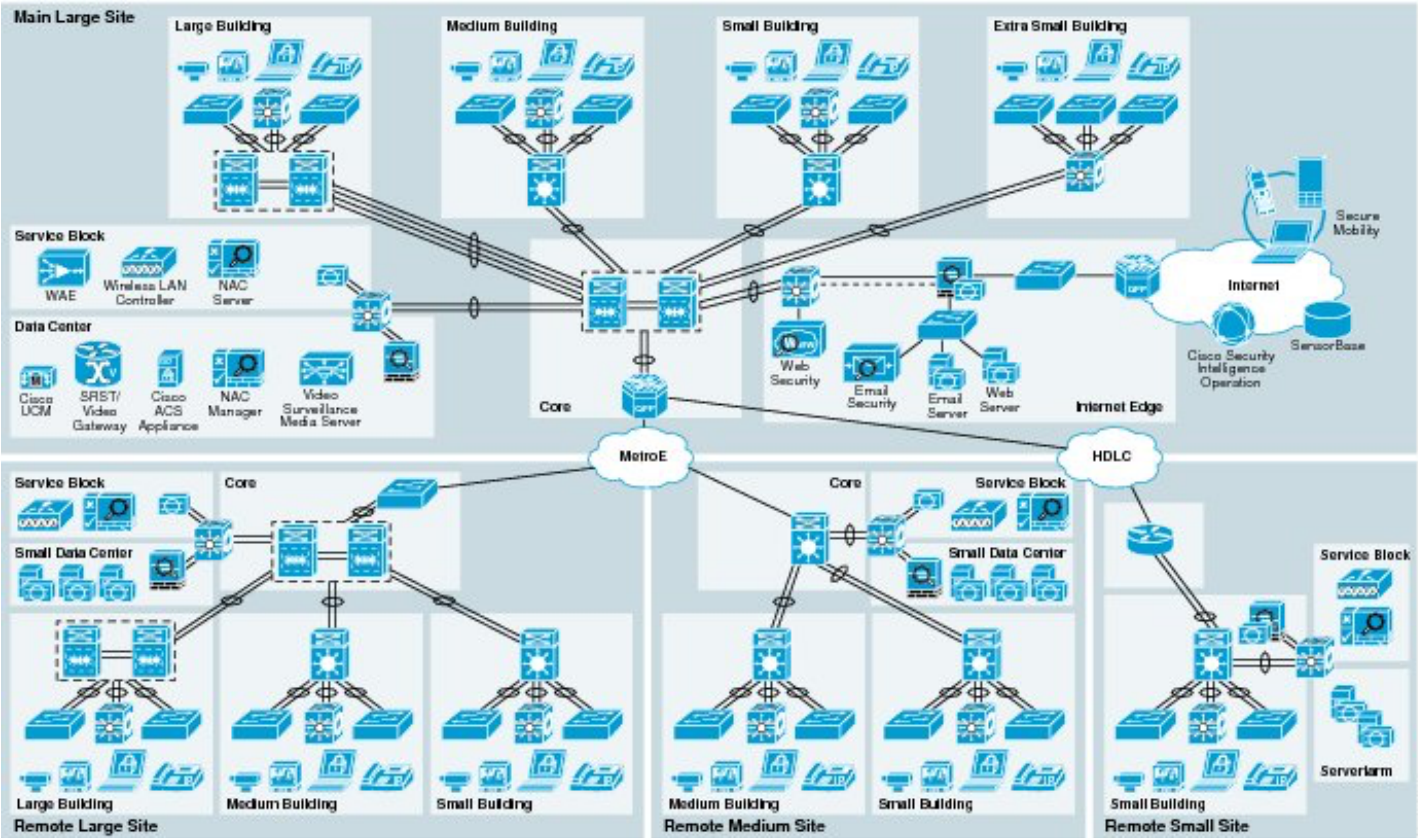
Research landscape & directions

Network verification in the real world

NETWORKING BACKGROUND

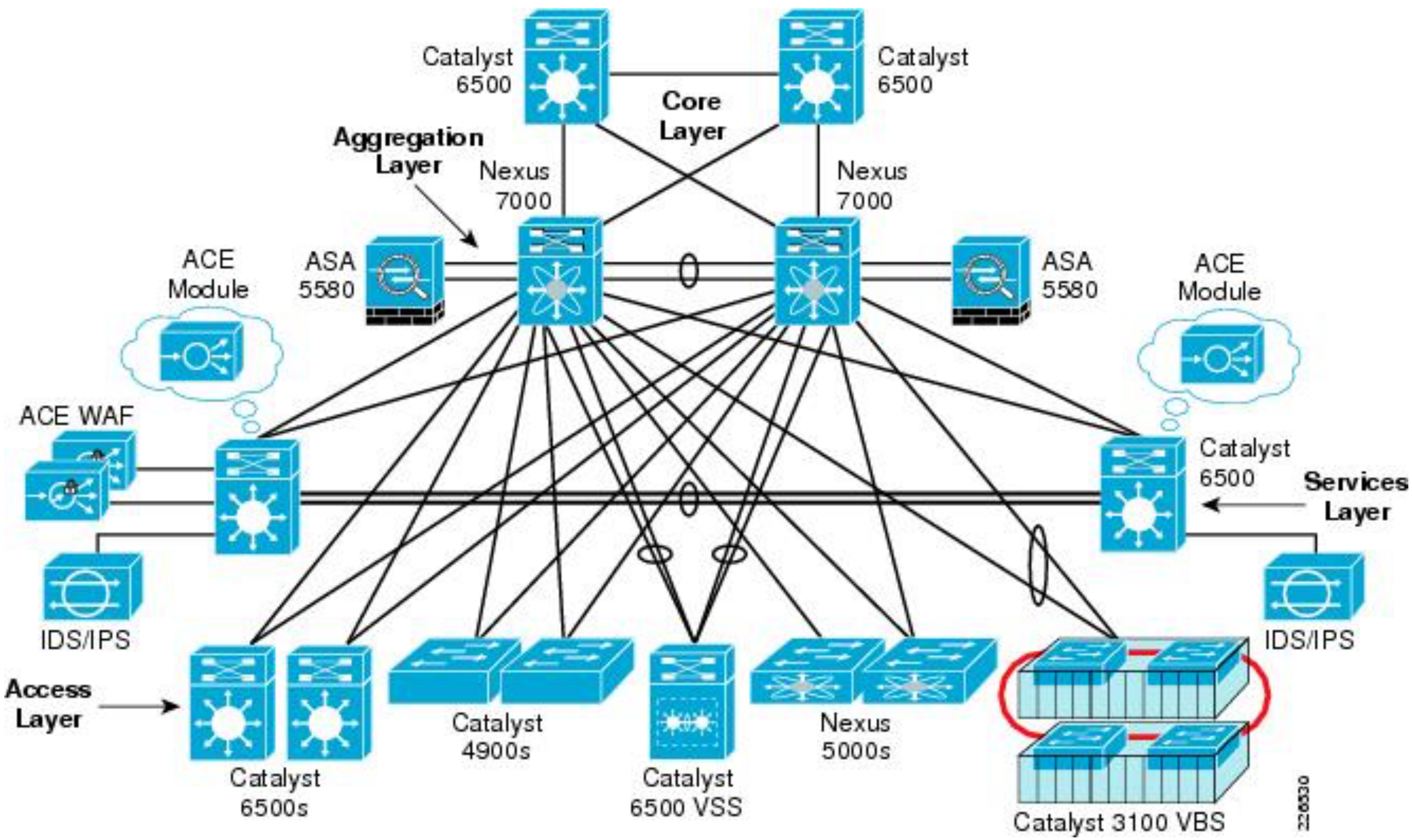


Inside a typical enterprise network



Source: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Medium_Enterprise_Design_Profile/MEDP/chap5.html

Inside a typical enterprise data center



Source: http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Security/SAFE_RG/SAFE_rg/chap4.html

Configs use many protocols & features

Layer 1 protocols (physical layer)

USB Physical layer
Ethernet physical layer including 10 BASE T, 100 BASE T,100 BASE TX,100 BASE FX, 1000 BASE T and other variants
varieties of 802.11 Wi-Fi physical layers
DSL
ISDN
T1 and other T-carrier links
E1 and other E-carrier links
Bluetooth physical layer

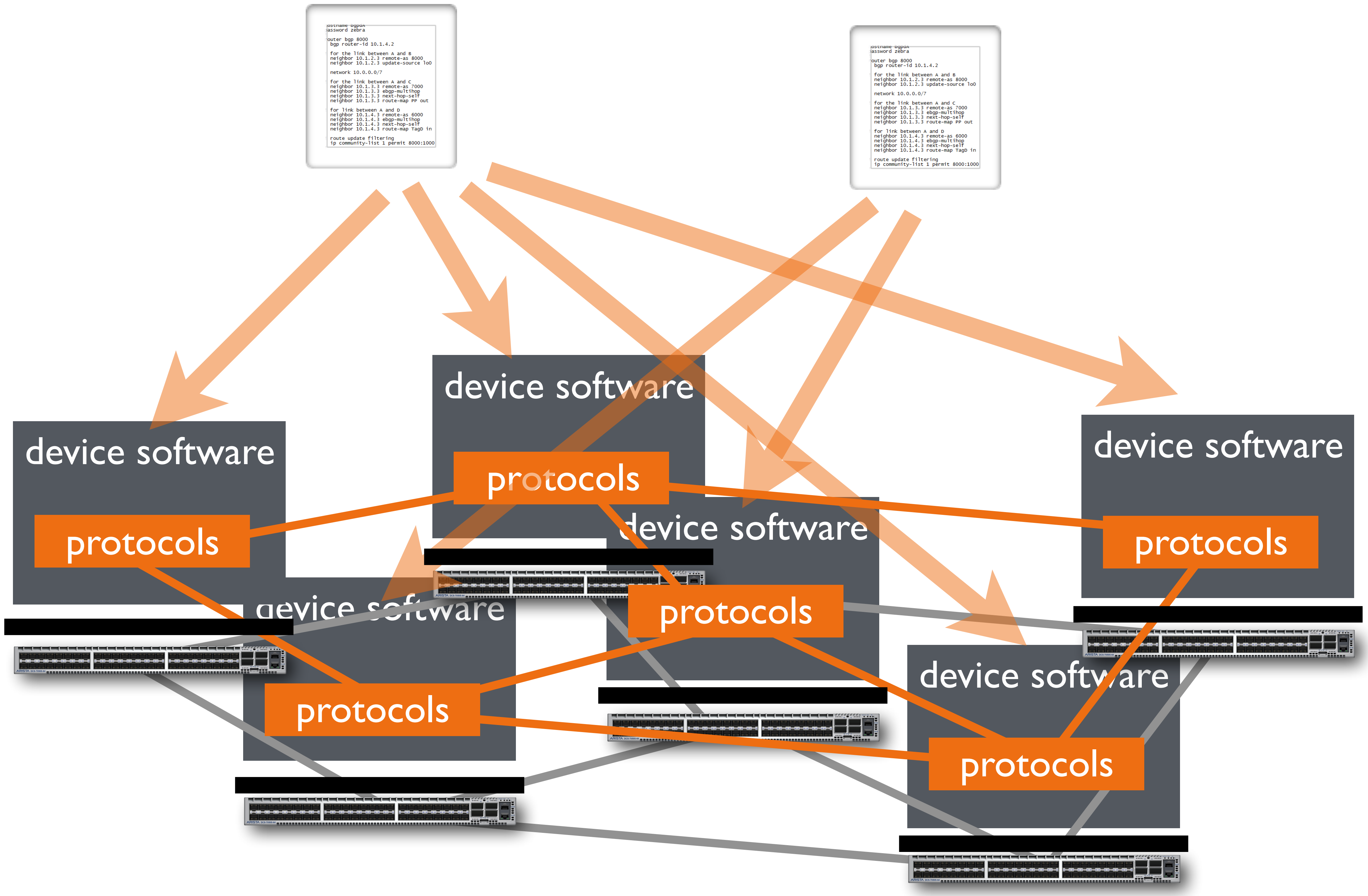
List of protocols commonly encountered by CCNAs
https://learningnetwork.cisco.com/docs/DOC-25649

Configs use many protocols & features

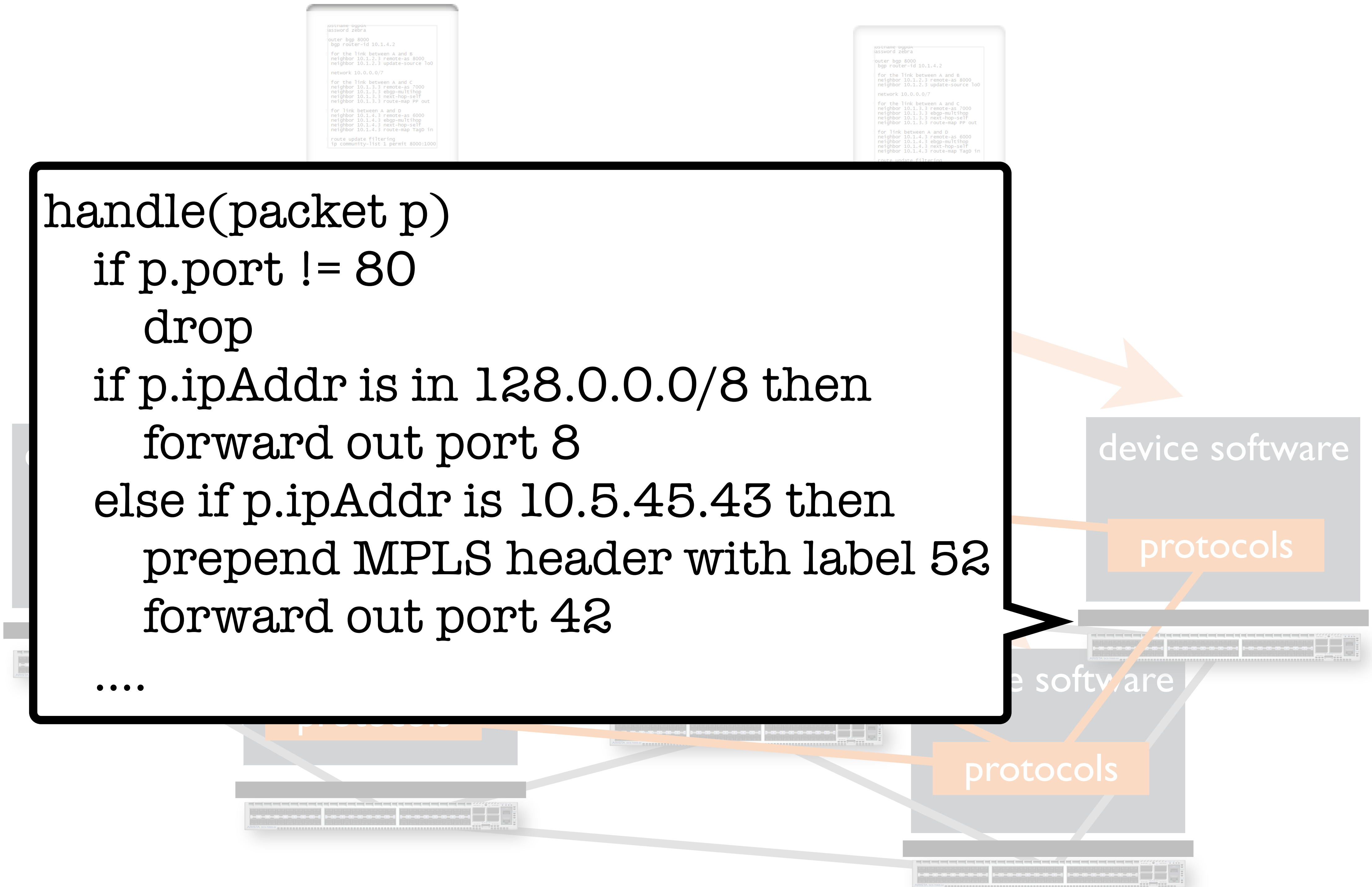
```
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname PrimaryR1
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
!
!
ip cef
!
interface Loopback100
no ip address
!
interface GigabitEthernet0/1
description LAN port
ip address 64.X.X.1 255.255.255.224
ip nat inside
ip virtual-reassembly
duplex auto
speed auto
media-type rj45
no negotiation auto
standby 1 ip 64.X.X.5
standby 1 priority 105
standby 1 preempt delay minimum 60
standby 1 track Serial3/0
!
interface GigabitEthernet0/2
description conn to Backup Lightpath
ip address 65.X.X.66 255.255.255.240
ip nat outside
ip virtual-reassembly
duplex full
speed 100
media-type rj45
no negotiation auto
!
interface GigabitEthernet0/3
description LAN handoff from P2P to Denver
ip address 10.30.0.1 255.254.0.0
duplex auto
speed auto
media-type rj45
no negotiation auto
!
interface Serial1/0
description p-2-p to Denver DC
ip address 10.10.10.1 255.255.255.252
dsu bandwidth 44210
framing c-bit
cablelength 10
clock source internal
serial restart-delay 0
!
interface Serial3/0
description DS3 XO WAN interface
ip address 65.X.X.254 255.255.255.252
ip access-group 150 in
encapsulation ppp
dsu bandwidth 44210
framing c-bit
cablelength 10
serial restart-delay 0
!
router bgp 16XX
no synchronization
bgp log-neighbor-changes
network 64.X.X.0 mask 255.255.255.224
network 64.X.X.2
aggregate-address 64.X.X.0 255.255.255.0 summary-only
neighbor 64.X.X.2 remote-as 16XX
neighbor 64.X.X.2 next-hop-self
neighbor 65.X.1X.253 remote-as 2828
neighbor 65.X.X.253 route-map setLocalpref in
neighbor 65.X.X.253 route-map localonly out
no auto-summary
!
no ip http server
!
ip as-path access-list 10 permit ^$
ip nat inside source list 101 interface GigabitEthernet0/2 overload
!
access-list 101 permit ip any any
access-list 150 permit ip any any
!
route-map setLocalpref permit 10
set local-preference 200
!
route-map localonly permit 10
match as-path 10
!
control-plane
!
gatekeeper
shutdown
!
!
end
```

Example basic BGP+HSRP config from https://www.myriadsupply.com/blog/?p=259

Distributed route computation



Result: data plane state

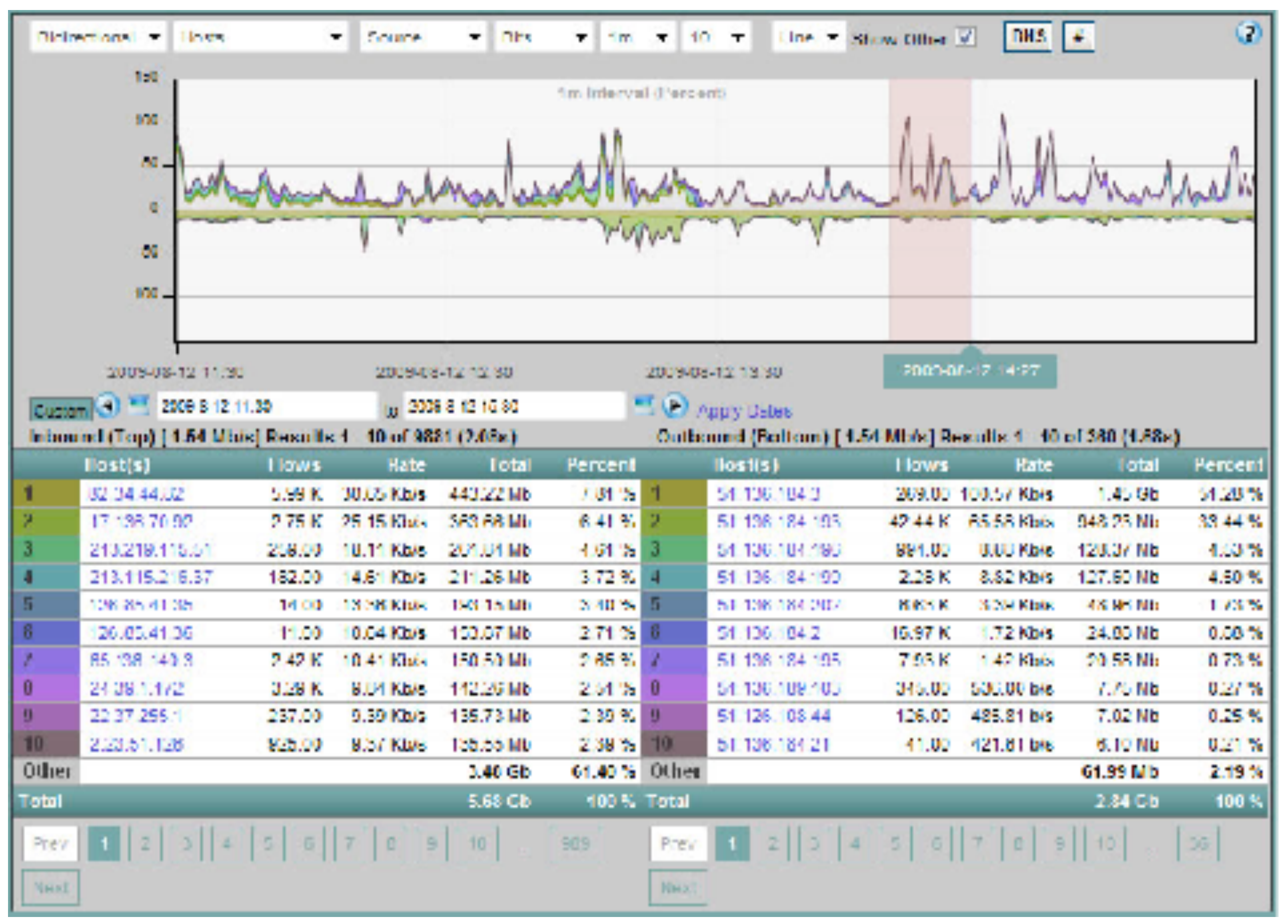



```
handle(packet p)
    if p.port != 80
        drop
    if p.ipAddr is in 128.0.0.0/8 then
        forward out port 8
    else if p.ipAddr is 10.5.45.43 then
        prepend MPLS header with label 52
        forward out port 42
    ....
```

Ensuring correct operations today

Manual spot-checking (pings, traceroutes)

Monitoring of events & flows



Screenshot from Scrutinizer
NetFlow & sFlow analyzer,
snmp.co.uk/scrutinizer/

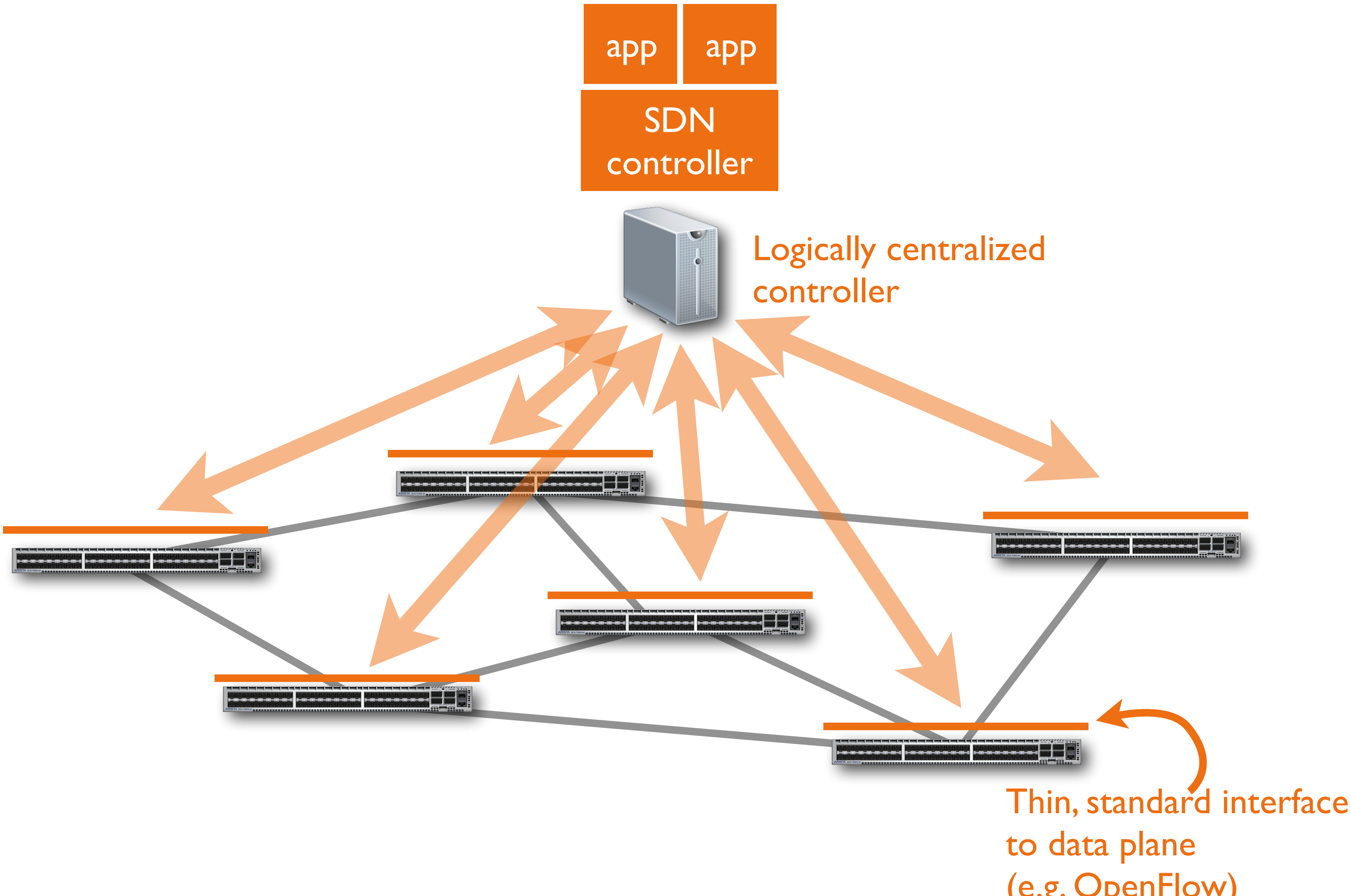
Networks are complex

89%
of operators never sure that config changes are bug-free

82%
concerned that changes would cause problems with existing functionality

– Survey of network operators
[Kim, Reich, Gupta, Shahbaz, Feamster, Clark, USENIX NSDI 2015]

Software-Defined Networks



Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

Network Verification

The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

"Host A should be connected to host B."

"Host A should not be able to reach service B on any server."

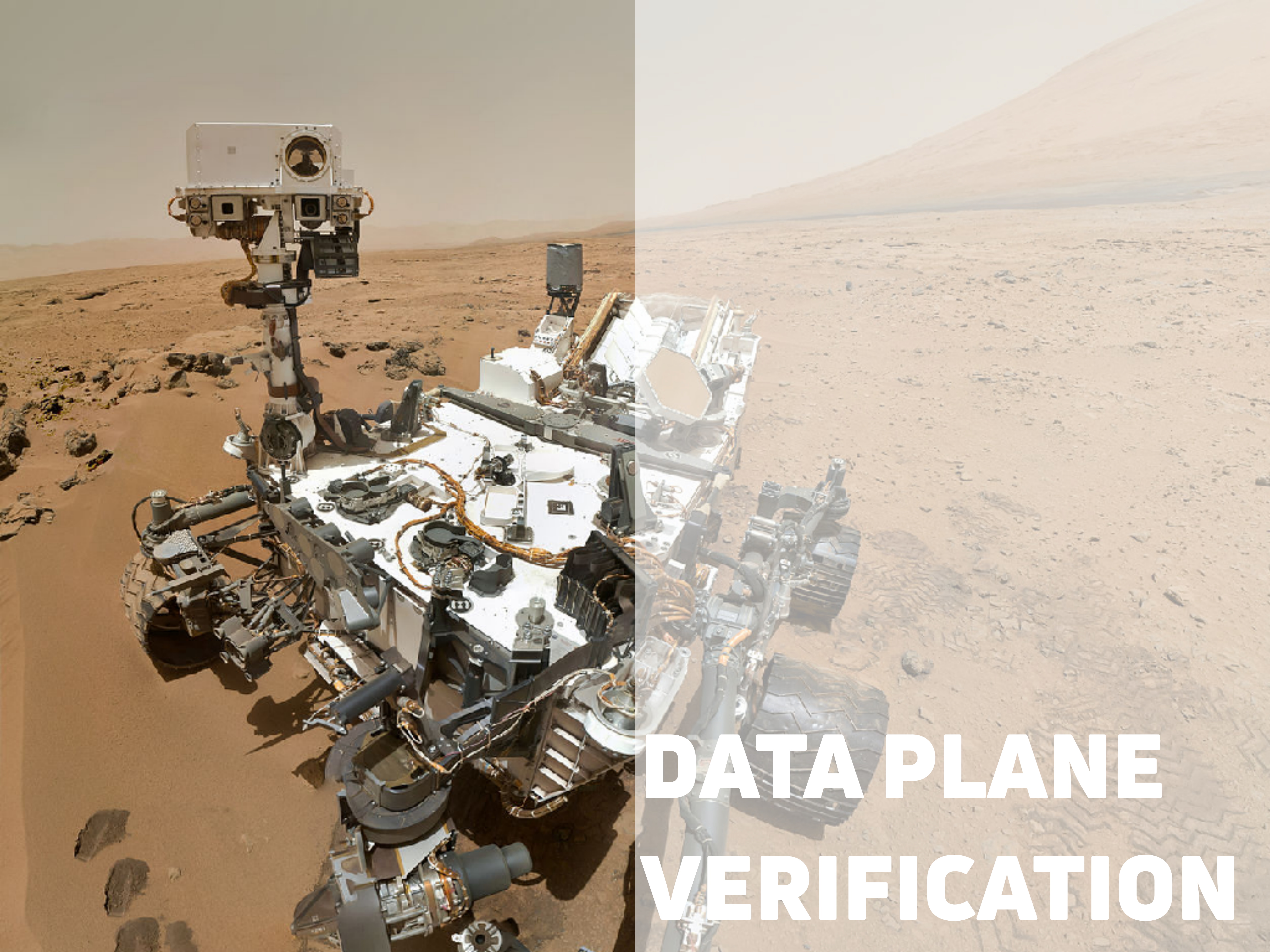
"No packet should fall into a loop."

"All packets should follow shortest paths."

Network Verification

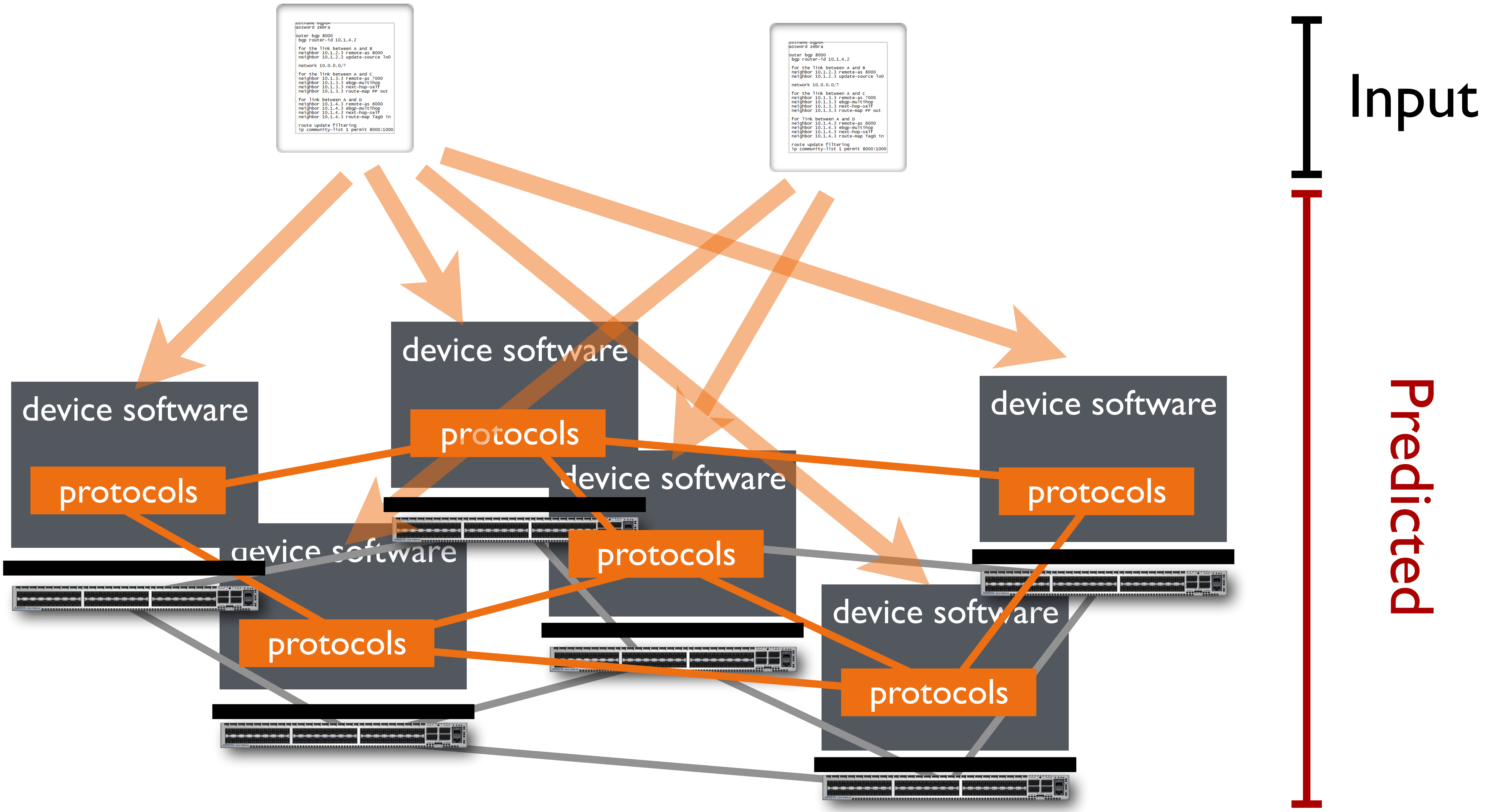
The process of proving whether an **abstraction** of the network satisfies intended network-wide **properties**.

Configuration	Configuration verification
Control software	Controller verification & verifiable control languages
Data plane state	Data plane verification
Packet processing	Software switch verification



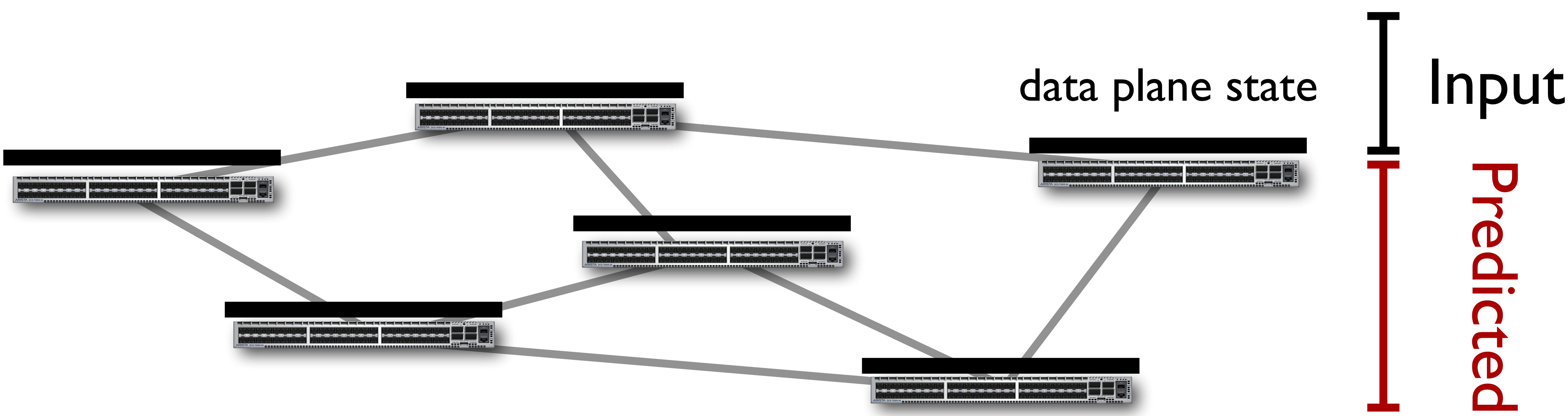
DATA PLANE VERIFICATION

Configuration verification



Data plane verification

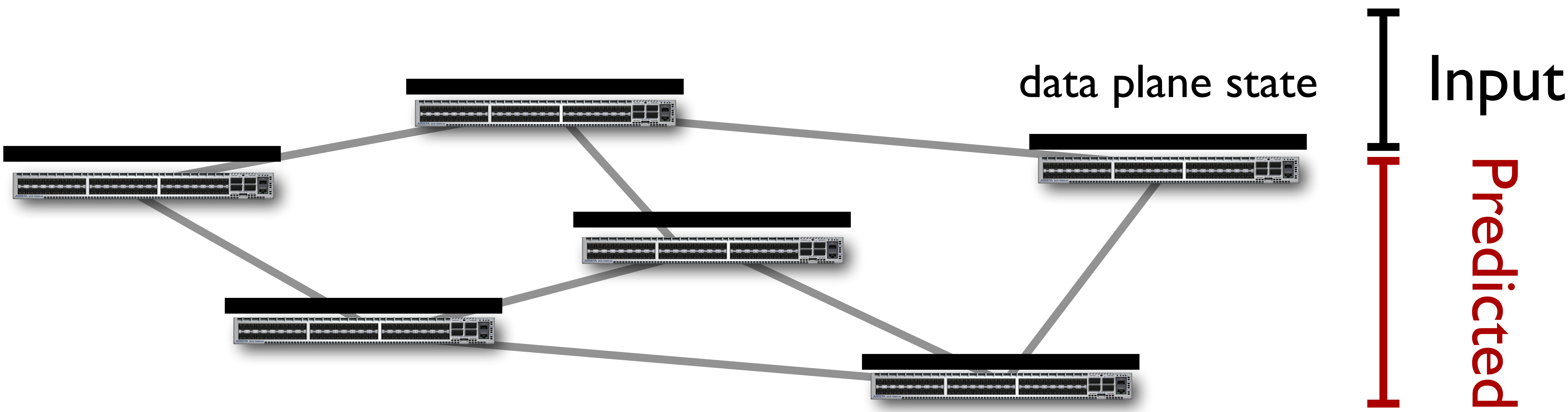
Verify the network
as close as possible
to its actual behavior



Data plane verification

Verify the network as close as possible to its actual behavior

- Insensitive to control protocols
- Accurate model
- *Checks current snapshot*



Need for accuracy

78 bugs sampled randomly from Bugzilla repository of Quagga (open source software router)

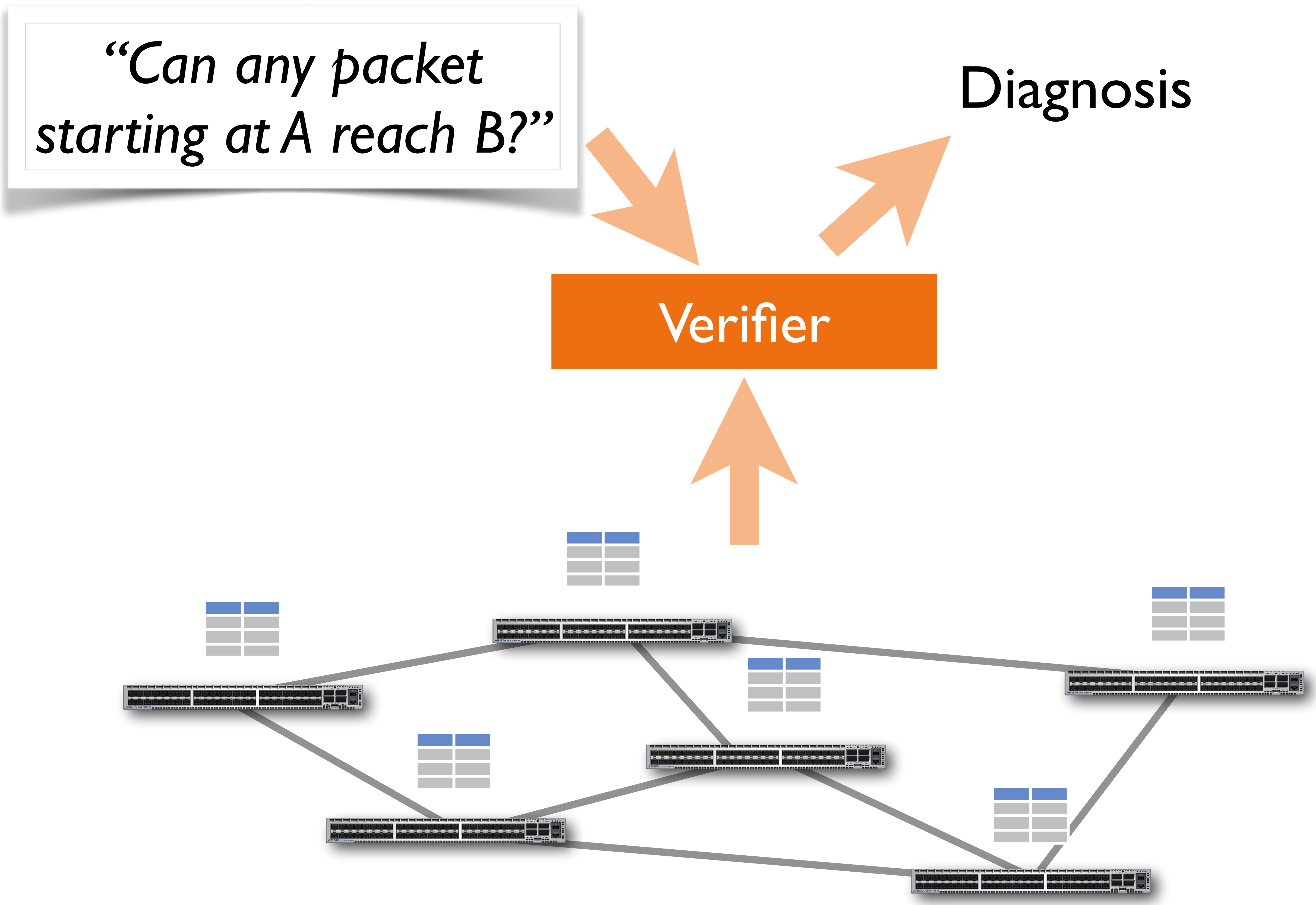
67 could cause data plane effect

- Under heavy load, Quagga 0.96.5 fails to update Linux kernel's routing tables
- In Quagga 0.99.5, a BGP session could remain active after it has been shut down

11 would not affect data plane

- Mgmt. terminal hangs in Quagga 0.96.4 on "show ip bgp"

Architecture



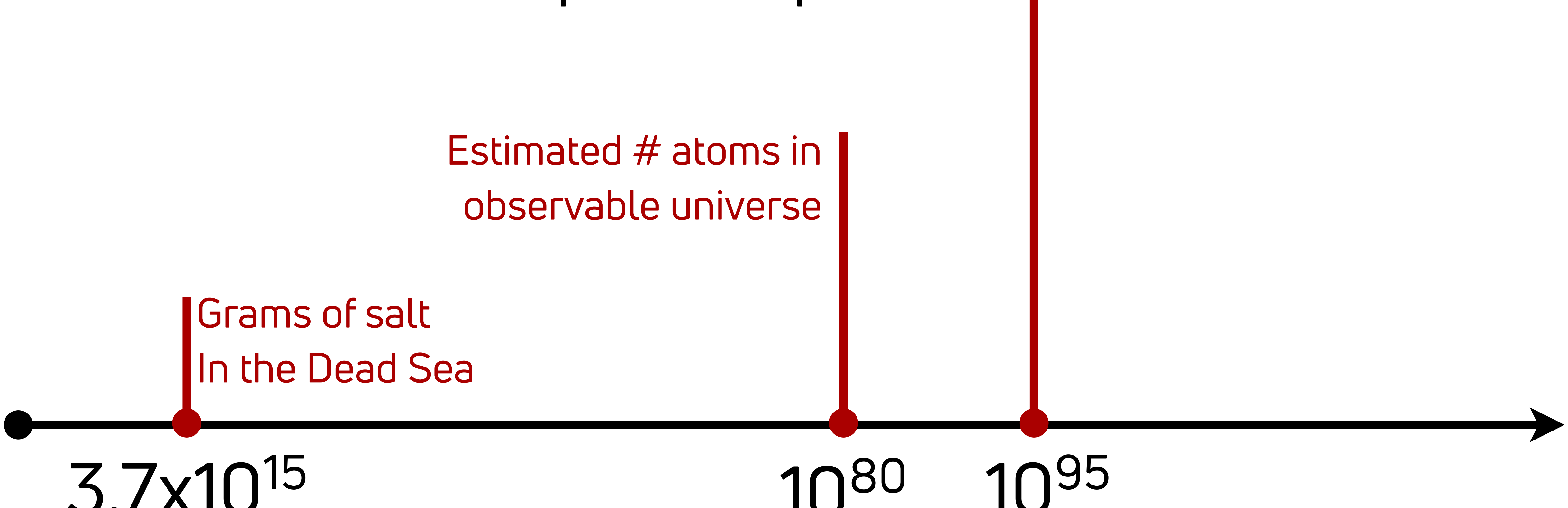
A little calculation...

\# theoretical packets

$= 2^{(\#\text{header bits})} \times \#\text{injection points}$

$= 2^{(18 \text{ byte ethernet } + 20 \text{ byte IPv4})} \times 10{,}000 \text{ ports}$

$= 3.25 \times 10^{95}$ possible packets



Digression into complexity theory

Given only **IP longest-prefix match** forwarding rules, how hard is it to compute whether A can reach B?

- (a) Polynomial time
- (b) NP-complete
- (c) Undecidable

…if we also allow **arbitrary bitmask** ("drop if bit 7 = 0")?

- NP-complete

…if we also allow **stateful devices** (e.g. firewall remembering connection establishment)?

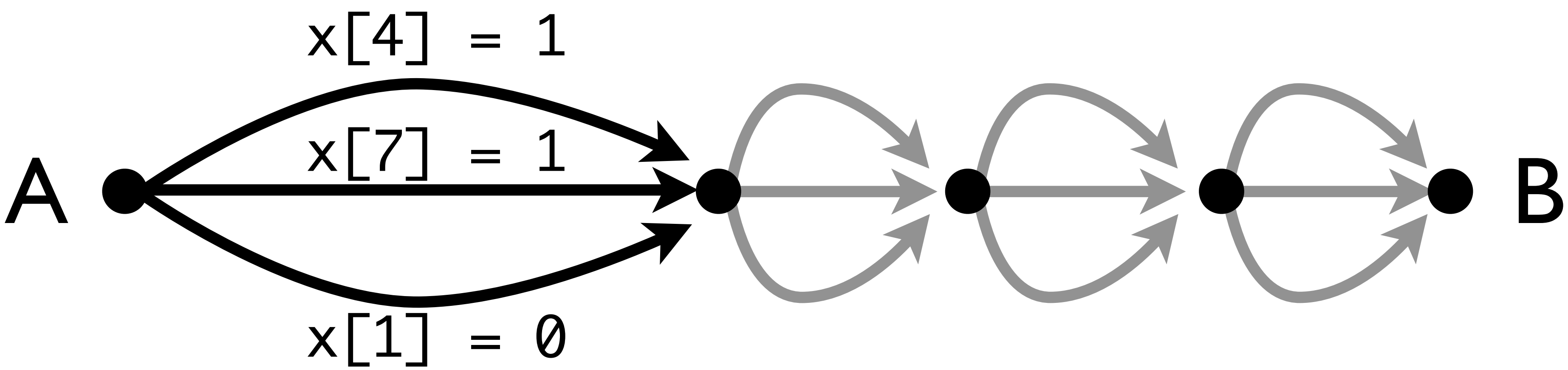
- Undecidable in general
- EXPSPACE-complete with reasonable assumptions
- Easier with additional assumptions

Some complexity results for stateful network verification

Velner, Alpernas, Panda, Rabinovich, Sagiv, Shenker, Shoham
TACACS 2016

A-to-B query with bitmask

Packet: x[0] x[1] x[2] ... x[n]



$$(x_4 \vee x_7 \vee \bar{x_1}) \wedge (\ldots) \wedge (\ldots) \wedge (\ldots)$$

NP-complete!

Anteater's solution

Express data plane and invariants as SAT

- …up to some max # hops
- Dynamic programming to deal with exponential number of paths
- Model packet transformations with vector of packet "versions" & constraints across versions

Check with off-the-shelf SAT solver (Boolector)

Debugging the Data Plane with Anteater
Mai, Khurshid, Agarwal, Caesar, Godfrey, King
SIGCOMM 2011

Data plane as boolean functions

Define P(u, v) as the expression for packets traveling from u to v

- A packet can flow over (u, v) *if and only if* it satisfies P(u, v)

Destination	Action
10.1.1.0/24	Fwd to v

$P(u, v) = dst\_ip \in 10.1.1.0/24$

Reachability as SAT solving

Goal: reachability from u to w

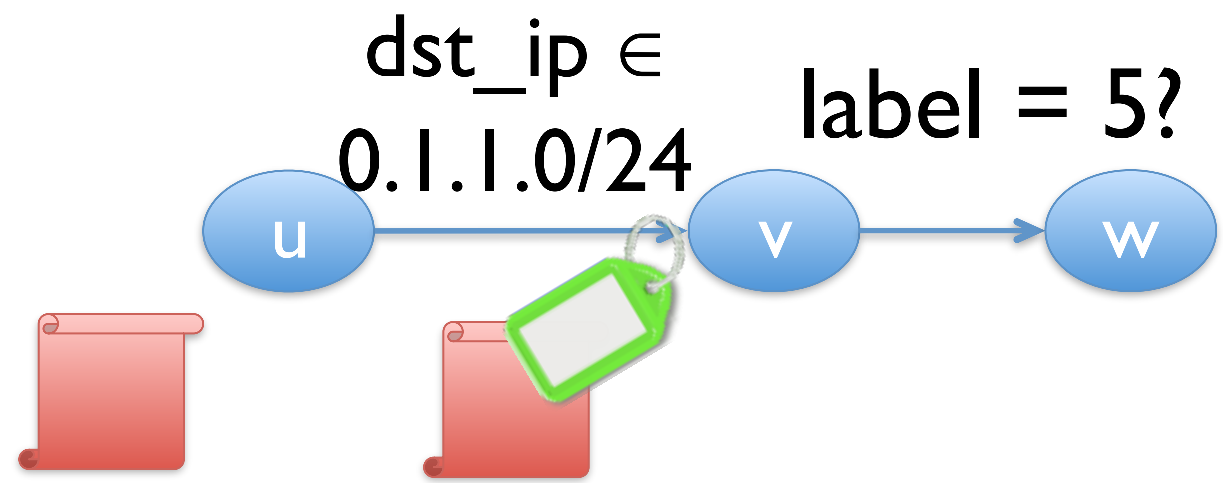
==

$C = (P(u, v) \wedge P(v,w))$ is satisfiable

- SAT solver determines the satisfiability of C
- Problem: exponentially many paths
  - Solution: Dynamic programming (a.k.a. loop unrolling)
  - Intermediate variables: “Can reach *x* in *k* hops?”
  - Similar to [Xie, Zhan, Maltz, Zhang, Greenberg, Hjalmtysson, Rexford, INFOCOM’05]

Packet transformation

Essential to model MPLS, QoS, NAT, etc.



- Model the history of packets: vector over time
- Packet transformation ⇒ boolean constraints over adjacent packet versions

$$(p_i.dst\_ip \in 0.1.1.0/24) \wedge (p_{i+1}.label = 5)$$

More generally: $p_{i+1} = f(p_i)$

Experiences with real network

Evaluated Anteater with operational network

- ∼178 routers supporting >70,000 machines
- Predominantly OSPF, also uses BGP and static routing
- 1,627 FIB entries per router (mean)
- State collected using operator's SNMP scripts

Revealed 23 violations of 3 invariants in 2 hours

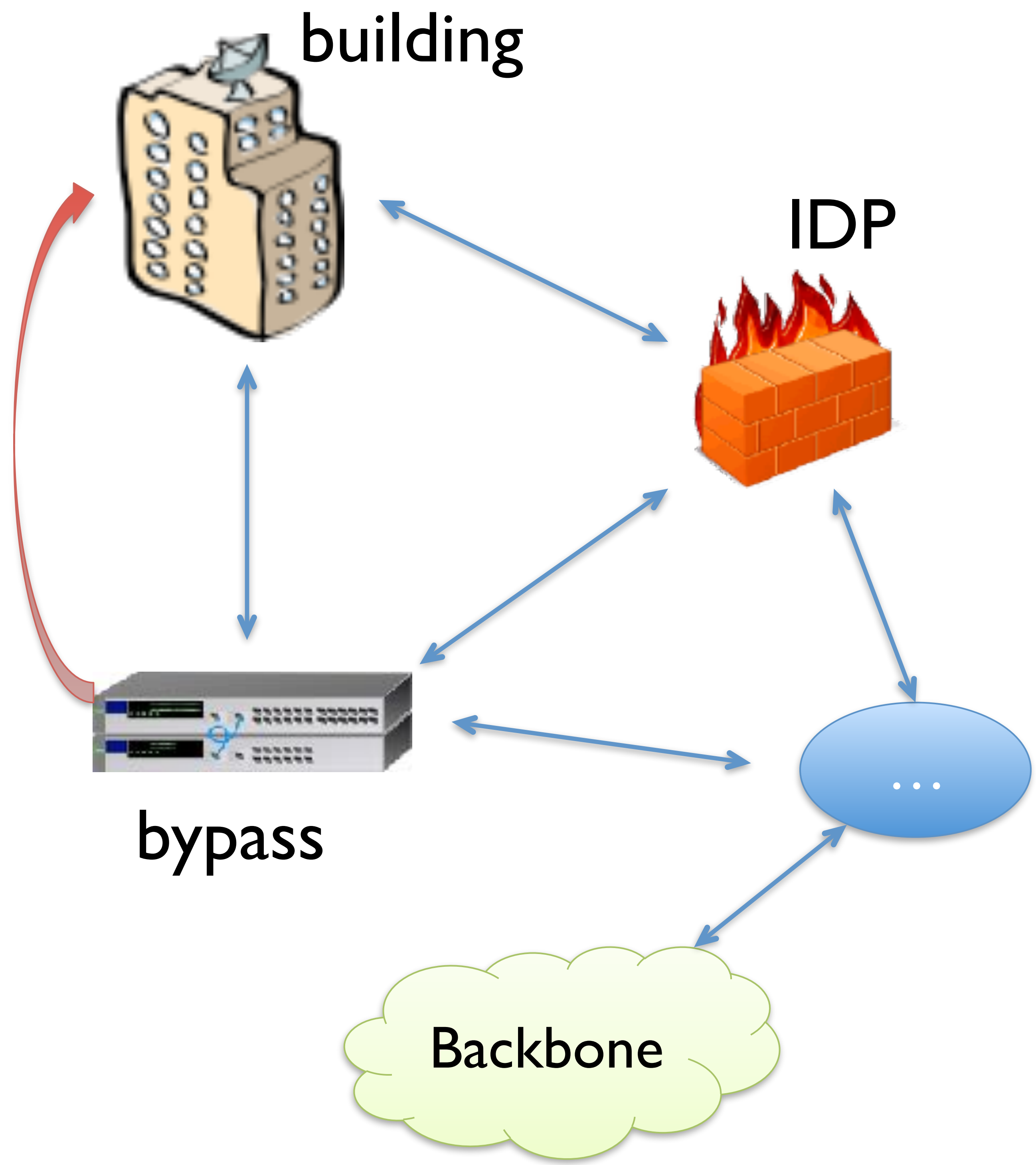
	Loop	Packet loss	Consistency
Being fixed	9	0	0
Stale config.	0	13	1
Total alerts	9	17	2

Forwarding loops

IDP was overloaded, operator introduced bypass

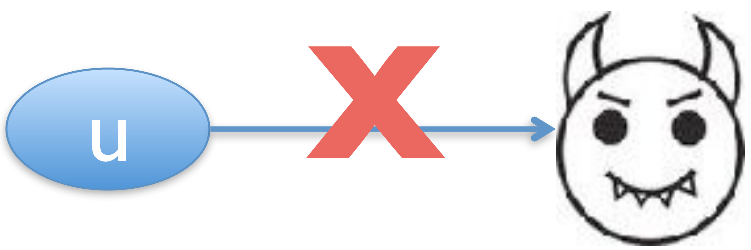
Bypass routed campus traffic to IDP through static routes

Introduced 9 loops



Multiple policy violations found

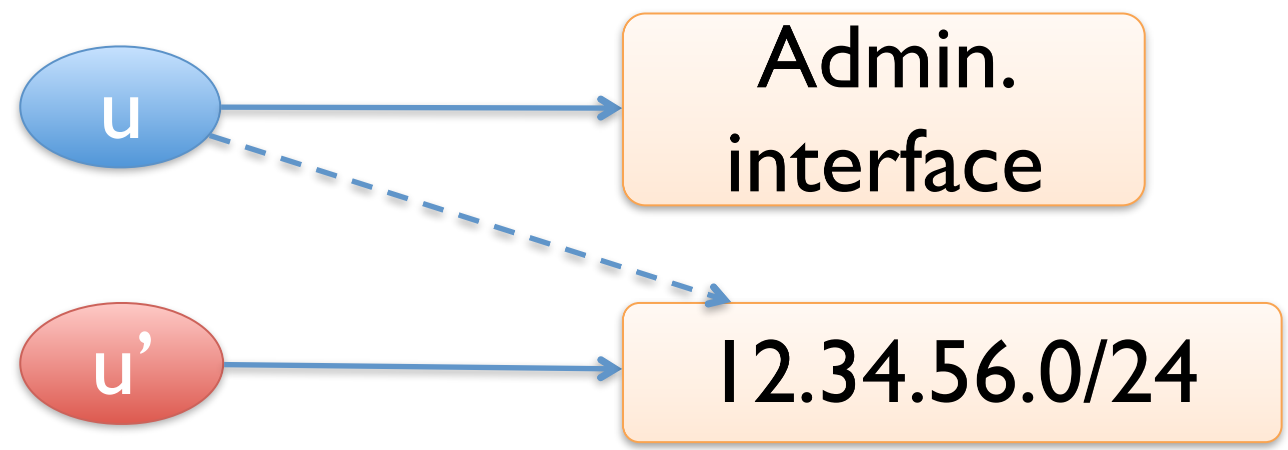
Packet loss



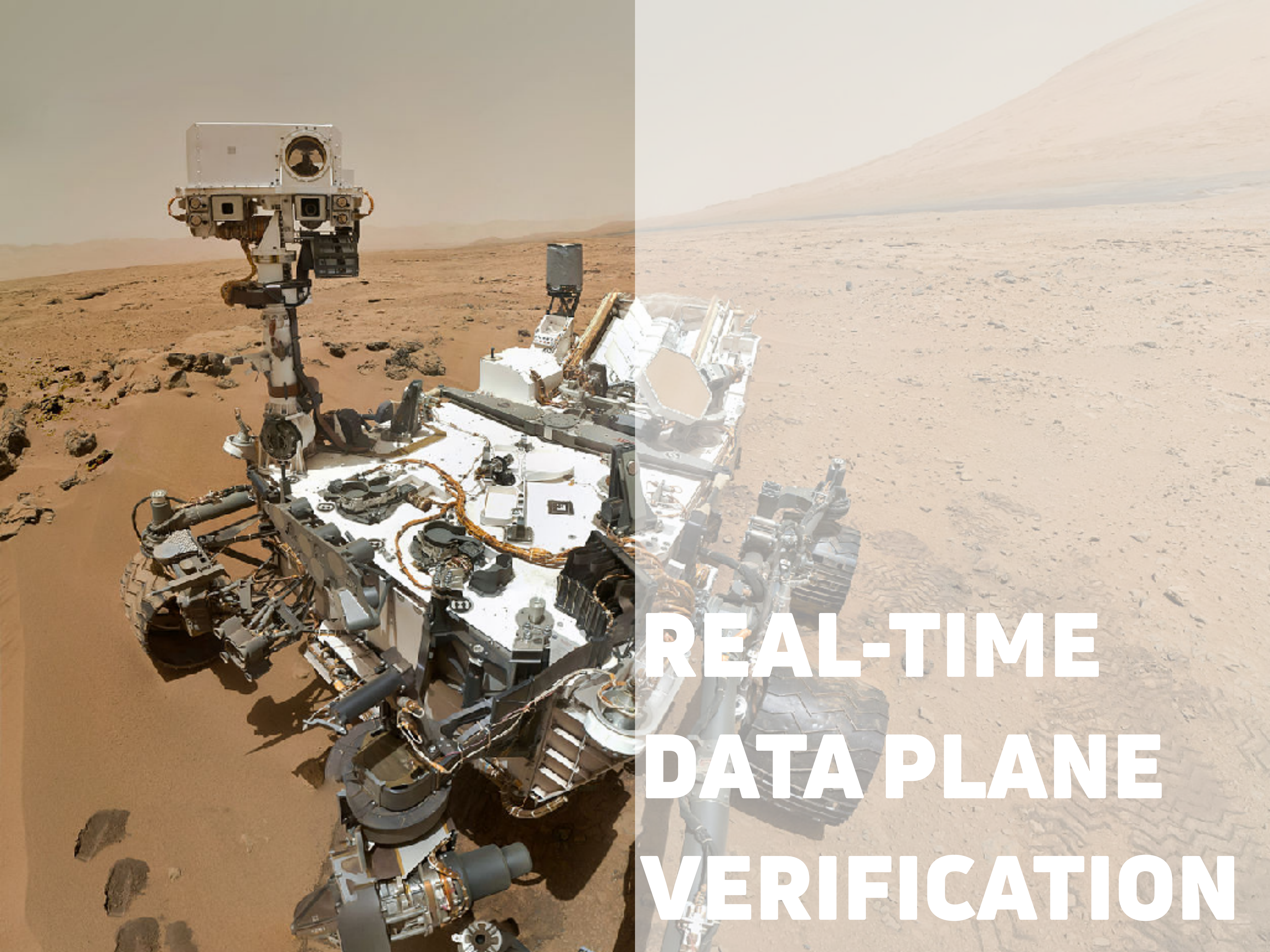
- Blocking compromised machines at IP level
- Stale configuration

From Sep, 2008

Consistency



- One router exposed web admin interface in FIB
- Different policy on private IP address range



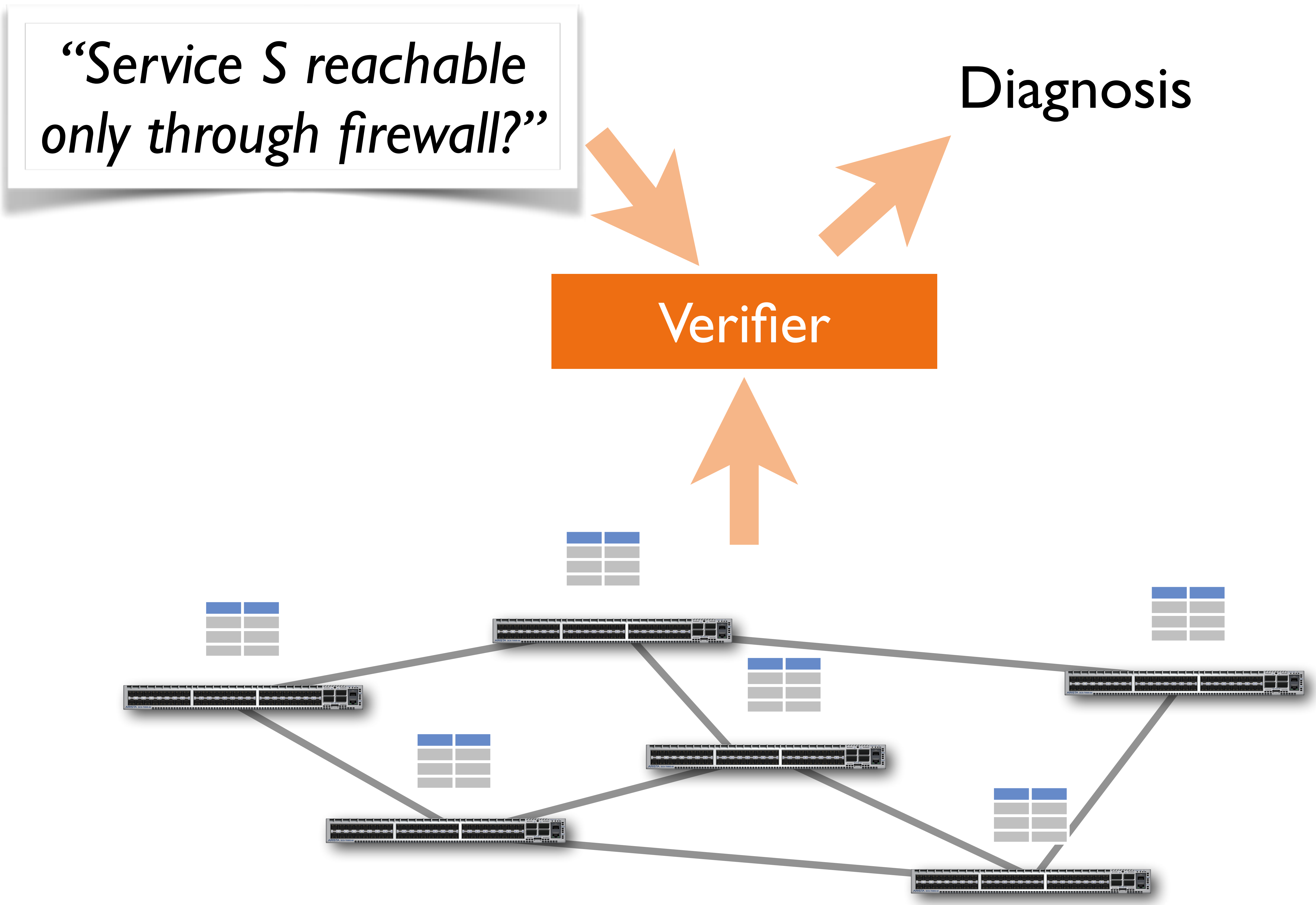
REAL-TIME DATA PLANE VERIFICATION

Not so simple

Challenge #1: Obtaining real time view

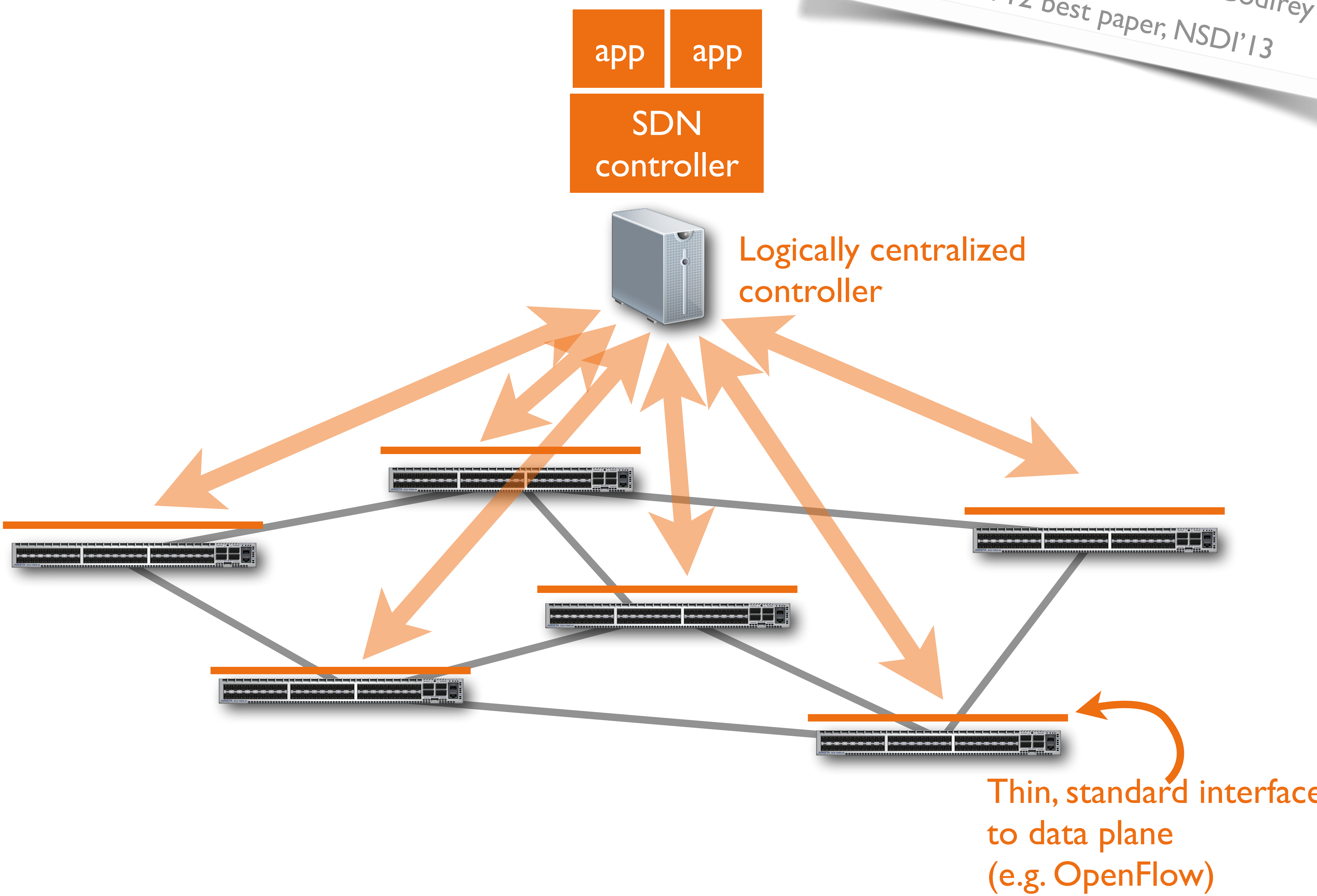
Challenge #2: Verify quickly

Architecture

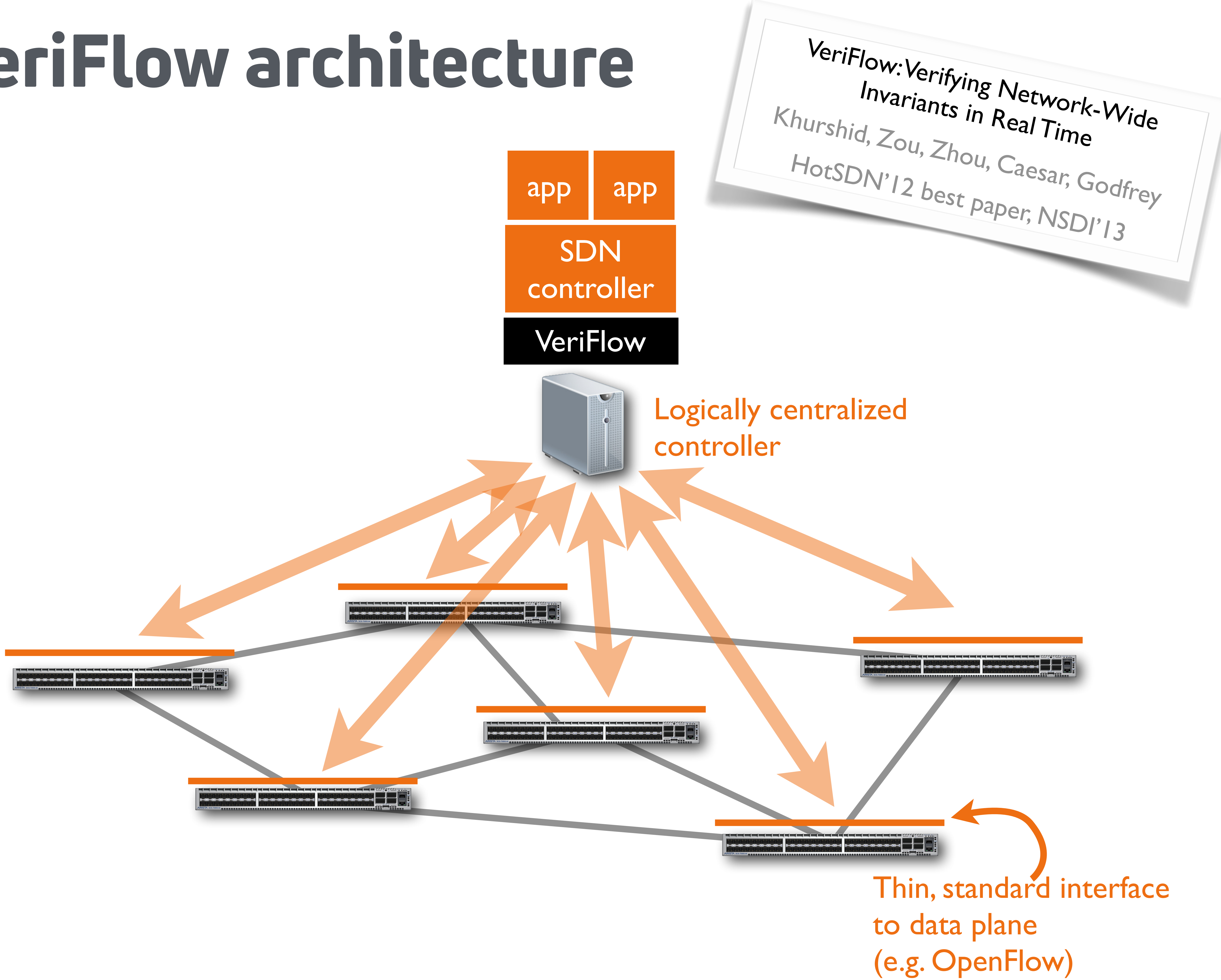


VeriFlow architecture

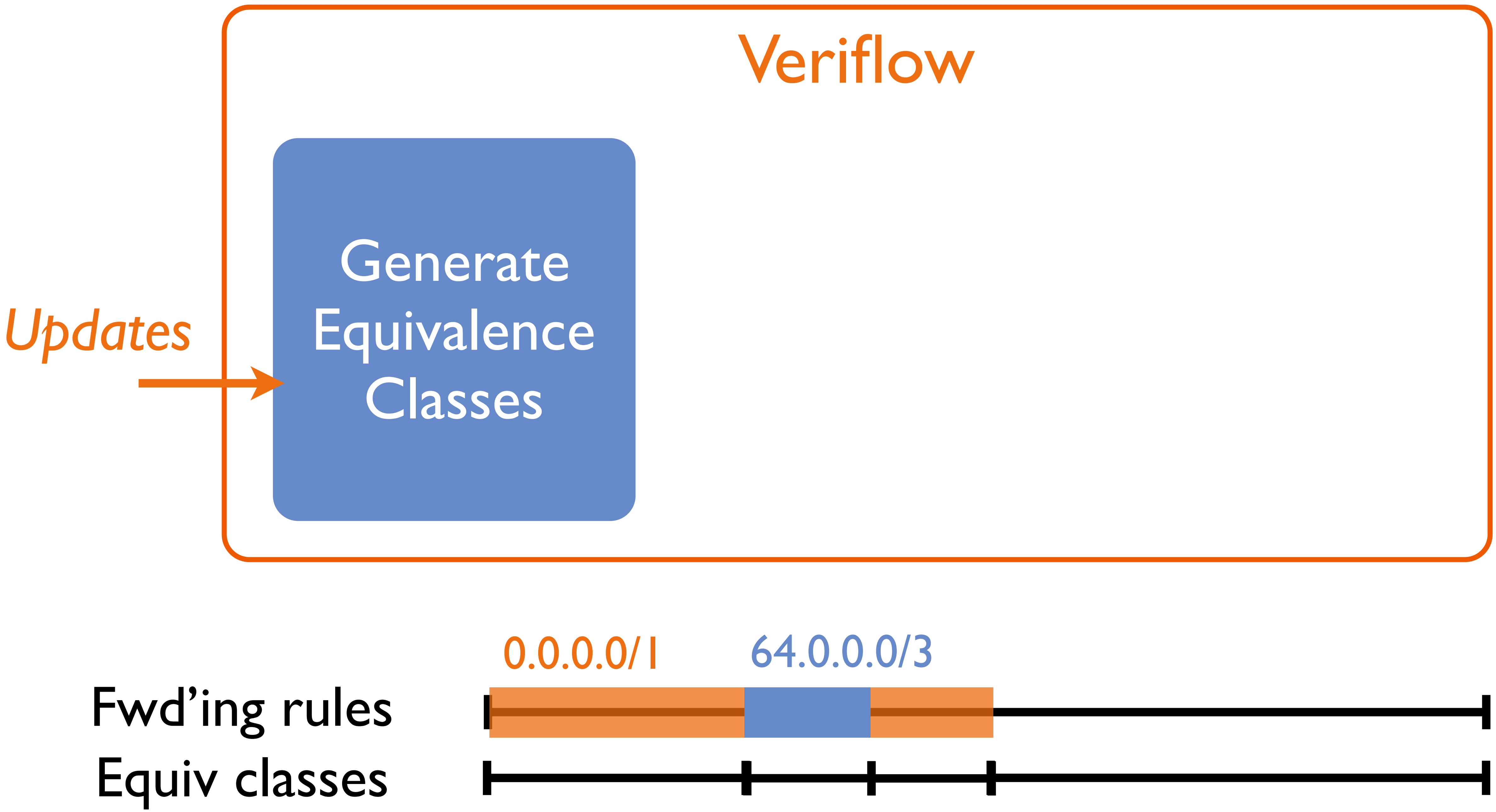
VeriFlow: Verifying Network-Wide Invariants in Real Time
Khurshid, Zou, Zhou, Caesar, Godfrey
HotSDN'12 best paper, NSDI'13



VeriFlow architecture

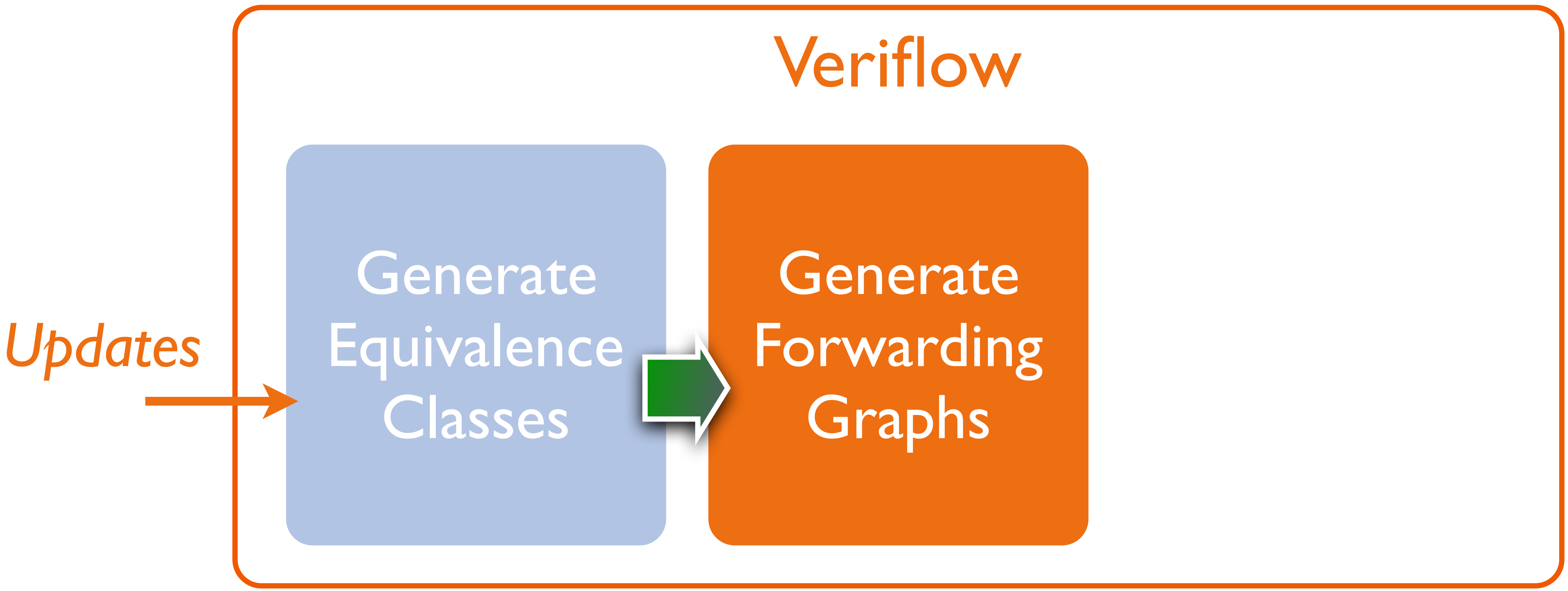


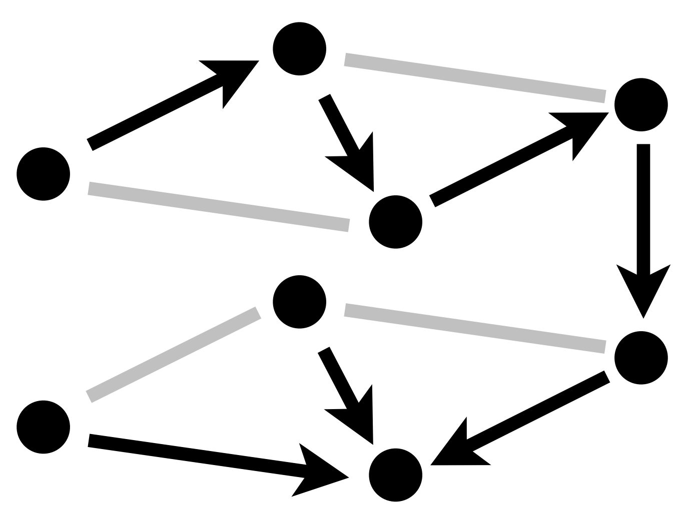
Verifying invariants quickly



Find only equivalence classes affected by the update via a multidimensional trie data structure

Verifying invariants quickly





All the info to answer queries!

Verifying invariants quickly

Verifiow

Updates

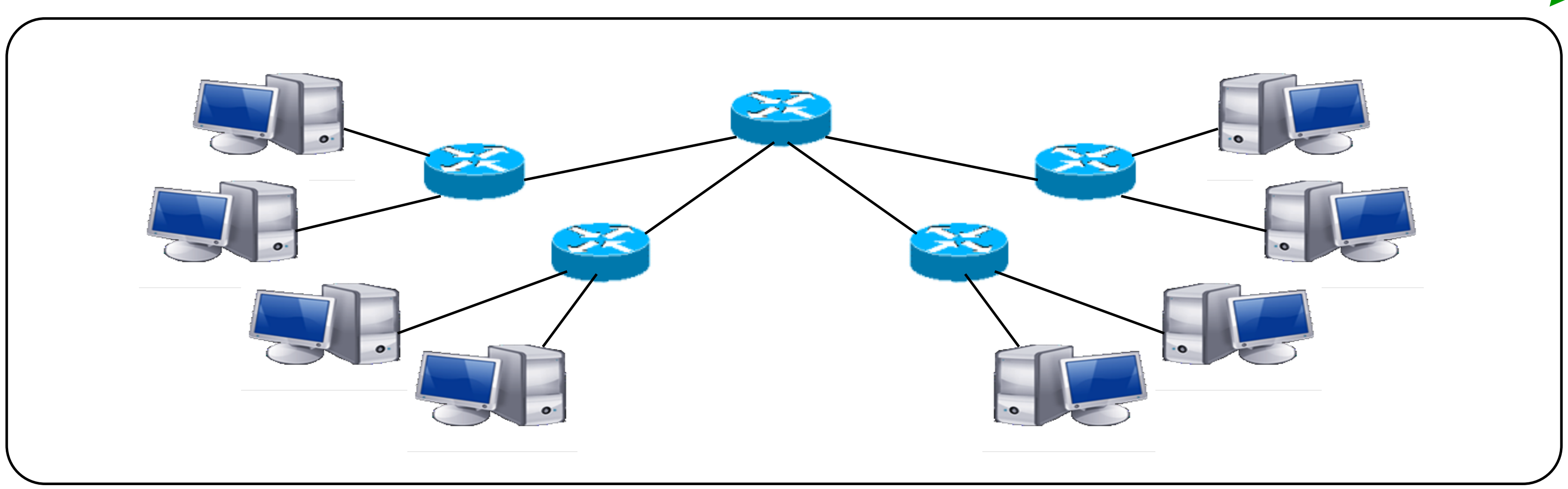
Generate Equivalence Classes

Generate Forwarding Graphs

Run Queries

Good rules

Bad rules



Diagnosis report

- Type of invariant violation
- Affected set of packets

Invariant API

Veriflow's API enables custom query algorithms

- Gives access to the "diff": equivalence classes and their forwarding graphs
- Verification becomes a standard graph traversal algorithm

What invariants can you check?

- Anything within data plane state (forwarding rules)...
- ...that can be verified *incrementally*

Evaluation

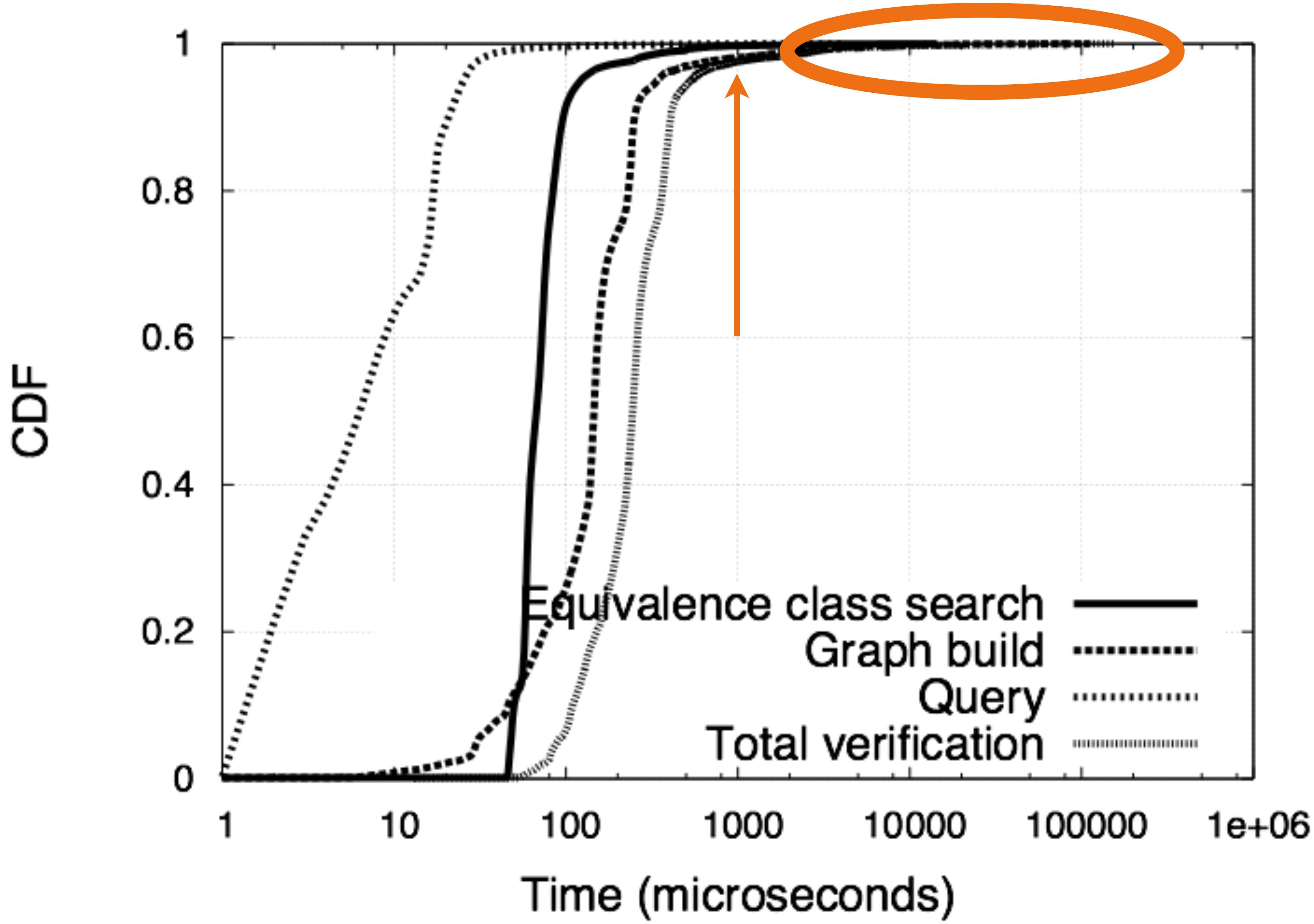
Simulated network

- Real-world BGP routing tables (RIBs) from RouteViews totaling 5 million RIB entries
- Injected into 172-router network (AS 1755 topology)

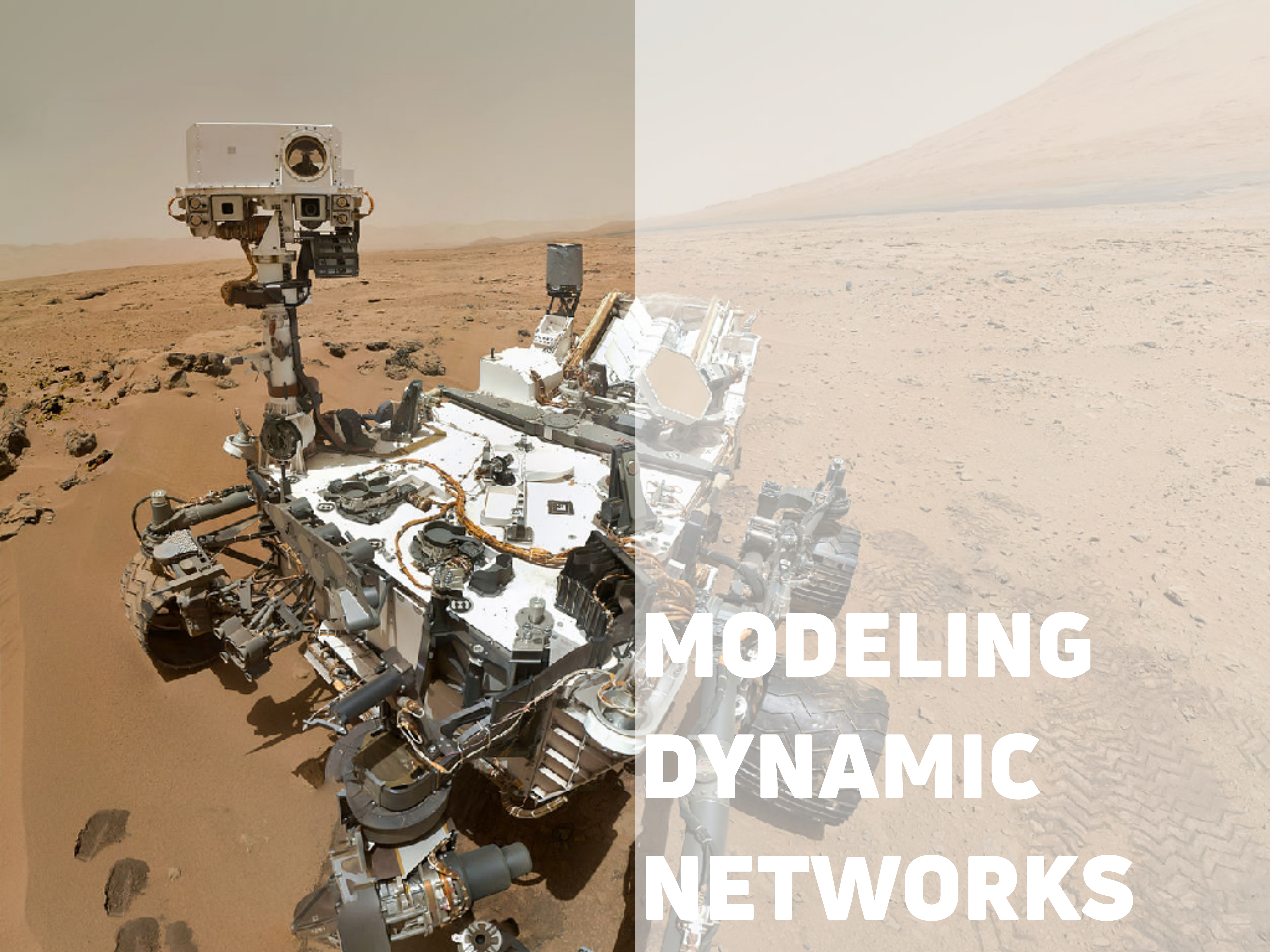
Measure time to process each forwarding change

- 90,000 updates from Route Views
- Check for loops and black holes

Microbenchmark latency

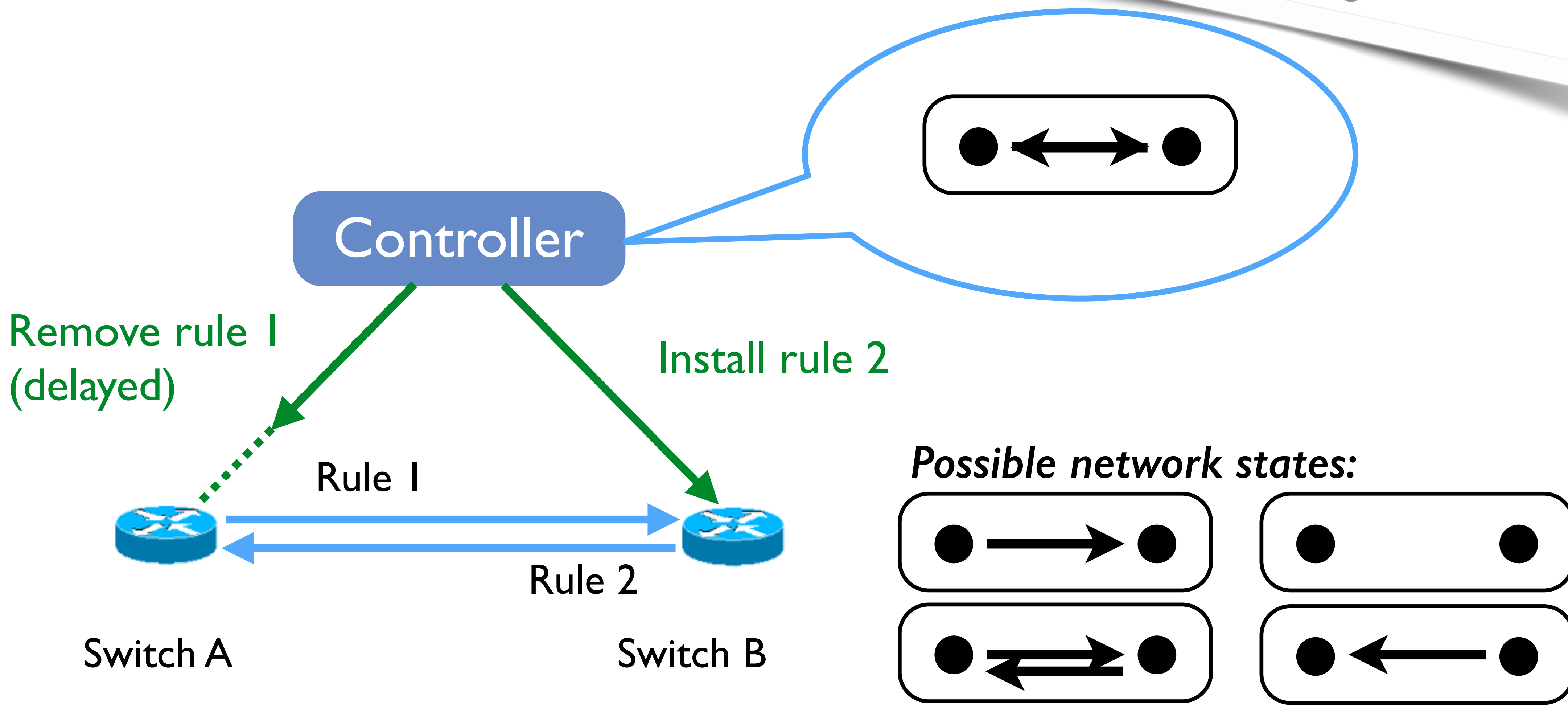


97.8% of updates verified within 1 ms


MODELING
DYNAMIC
NETWORKS

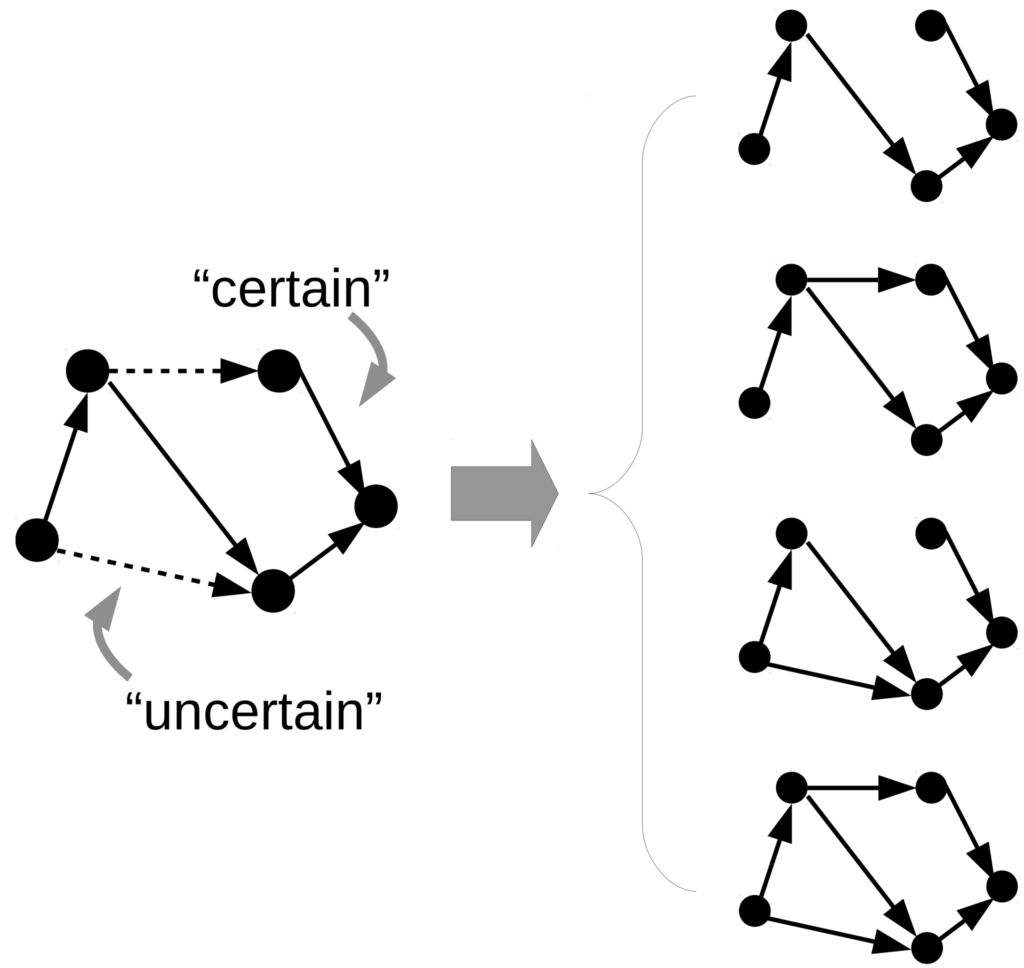
Timing uncertainty

Enforcing Customizable Consistency Properties in Software-Defined Networks
Zhou, Jin, Croft, Caesar, Godfrey
NSDI 2015

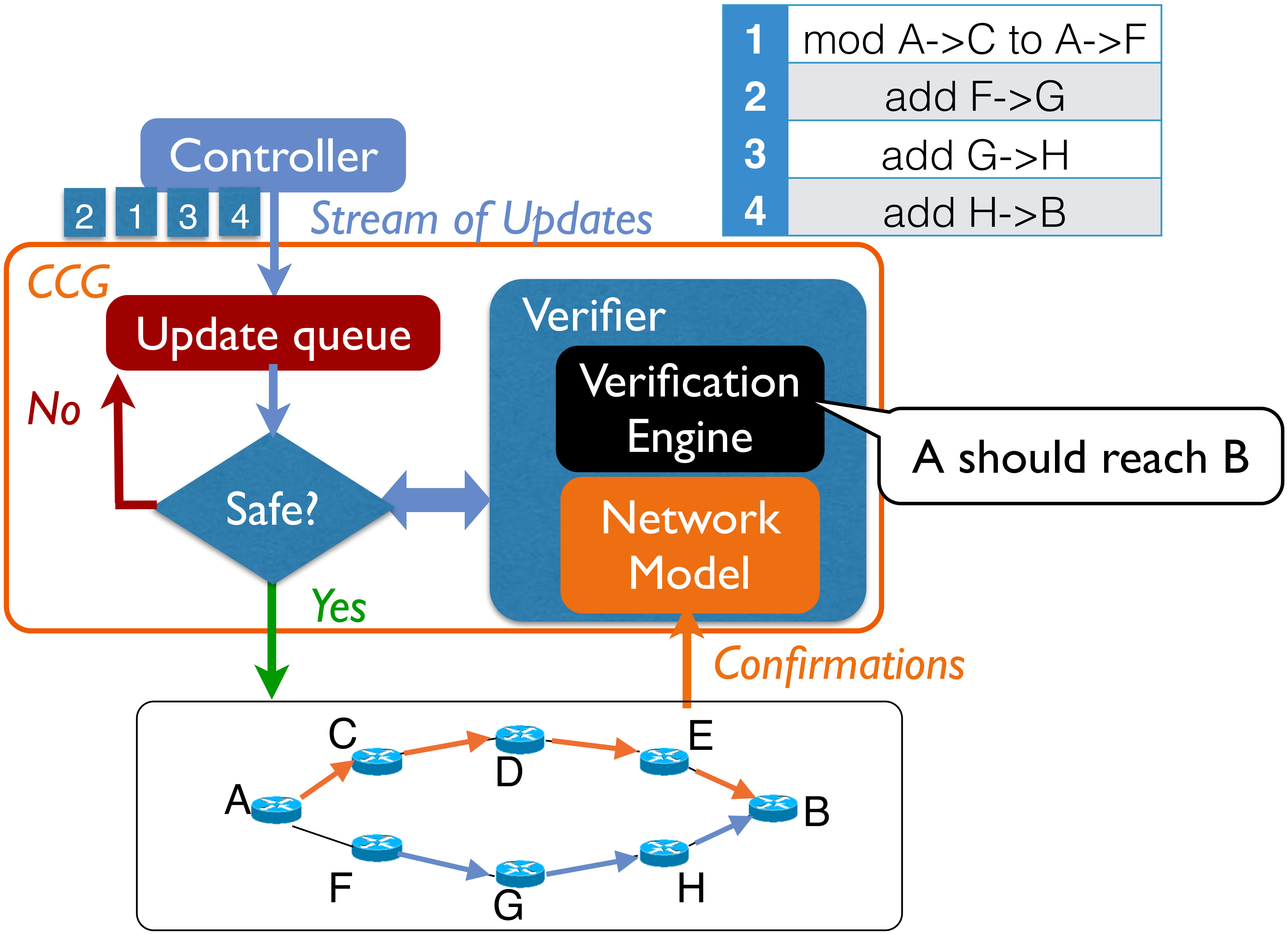


One solution: "consistent updates"
[Reitblatt, Foster, Rexford, Schlesinger, Walker, "Abstractions for Network Update", SIGCOMM 2012]

Uncertainty-aware verification



Update synthesis via verification



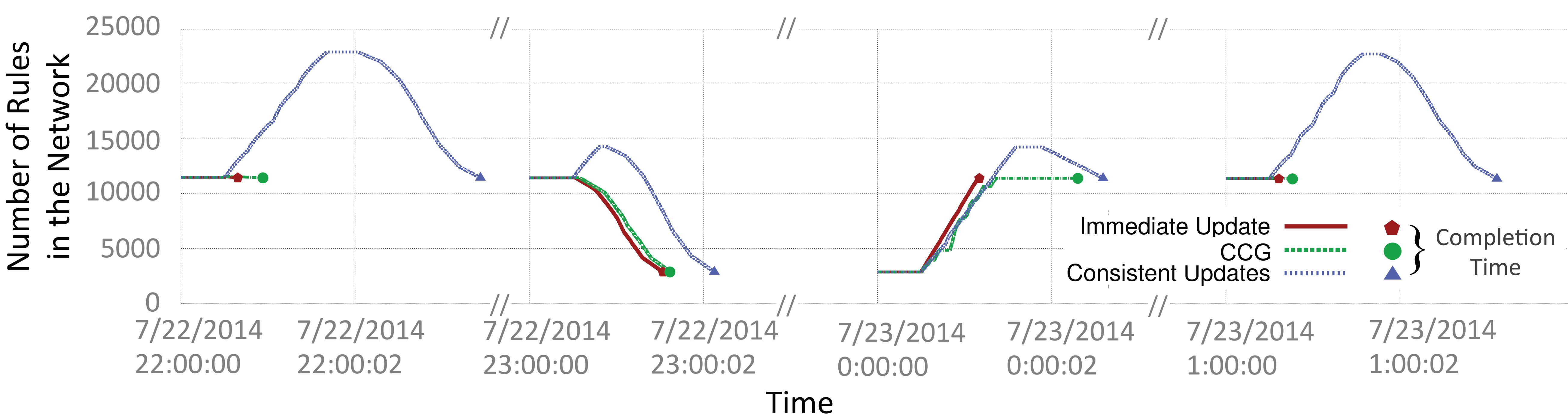
Enforcing dynamic correctness with heuristically maximized parallelism

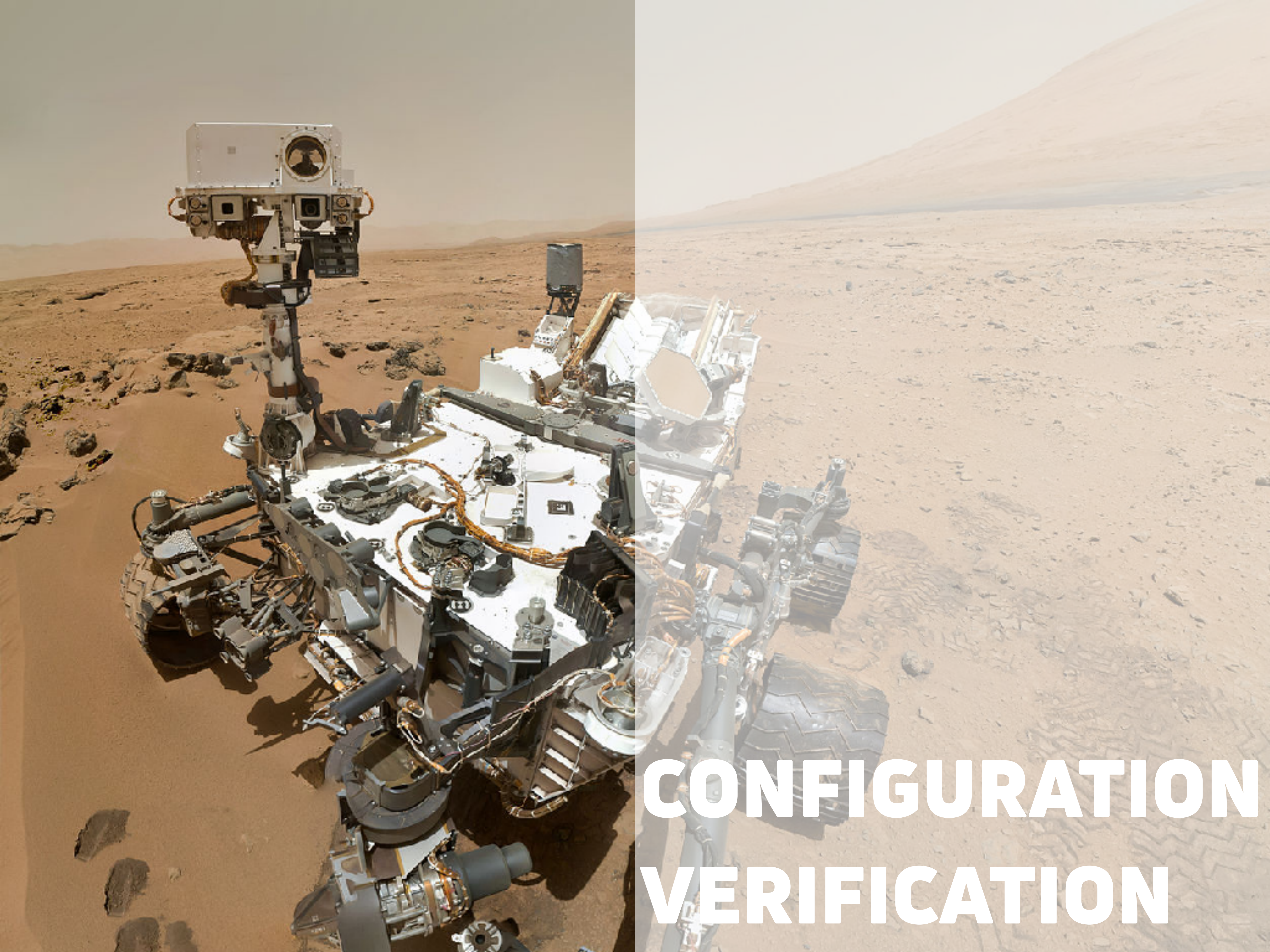
OK, but...

Can the system "deadlock"?

- Proved classes of networks that never deadlock
- Experimentally rare in practice!
- Last resort: heavyweight "fallback" like consistent updates [Reitblatt et al, SIGCOMM 2012]

Is it fast?




CONFIGURATION
VERIFICATION

Challenges and Approach

Slides in this section thanks to the Batfish team

A general approach to network configuration analysis

Fogel, Fung, Pedrosa, Walraed-Sullivan, Govindan, Mahajan, Millstein
NSDI 2015

Challenges in faithfully deriving the data plane

- Accurately model low-level configuration directives
- Provide high-level understanding of errors to operators

Approach: High-fidelity declarative model of control plane

- Set of relations that expresses the network’s control plane computation
- Provides queryability and provenance for free

Batfish

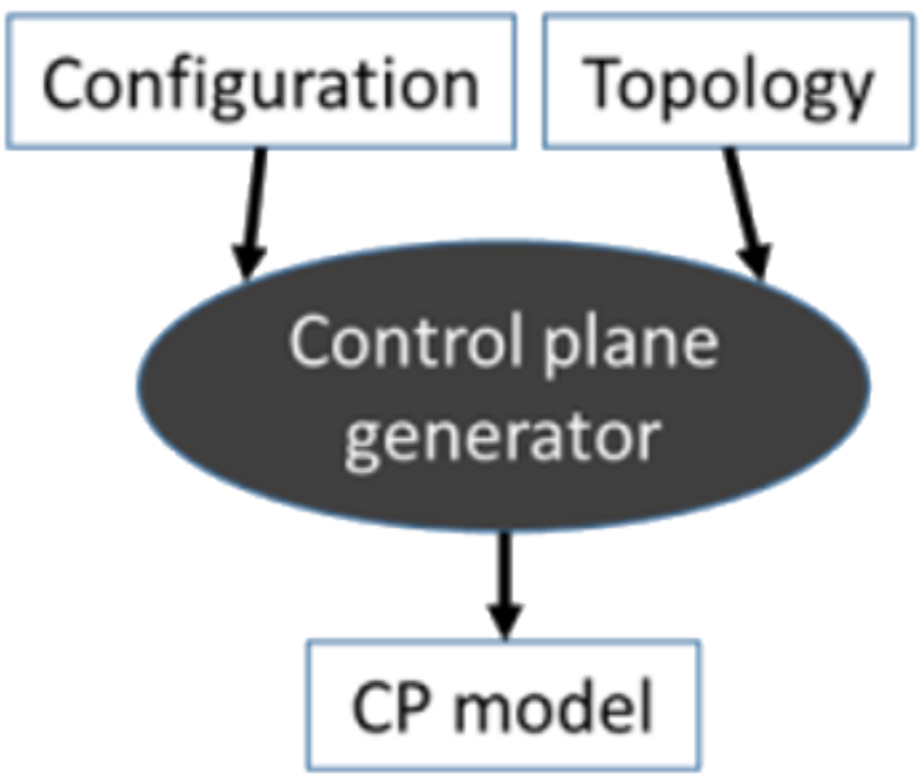
Available at http://www.batfish.org

Has found real bugs in real networks

4 stages:

- Control plane generator
- Data plane generator
- Safety analyzer
- Provenance tracker

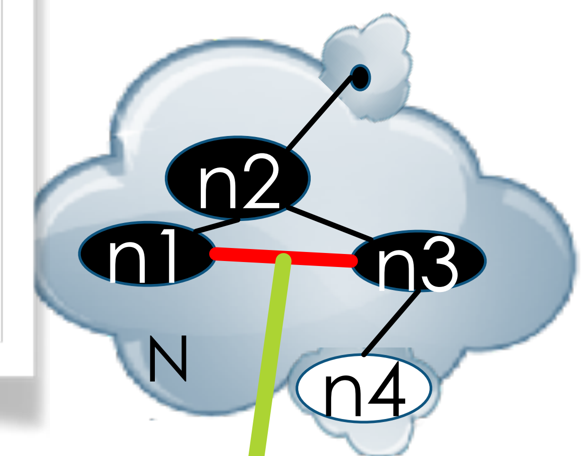
Stage 1: Extract control plane model



//----------Configuration of n3----------
1 ospf interface int3_1 metric 1
2 ospf interface int3_2 metric 1
3 ospf interface int3_4 metric 1

4 static route 10.0.0.0/24 drop
5 ospf redistribute static metric 10

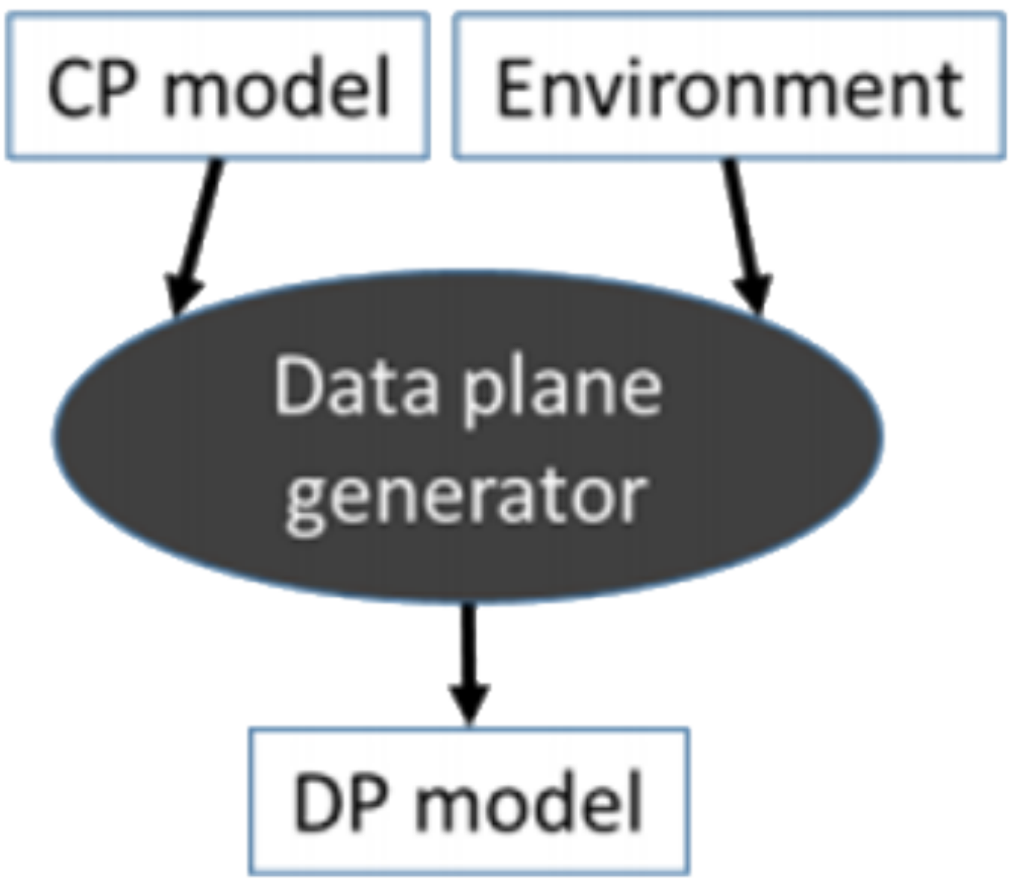
6 bgp neighbor p1 AS P Accept ALL



Fact about OSPF interface costs
OspfCost(
node:n3,
interface:int3_1,
cost:1).

Fact about topology
LanNeighbors(
node1:n3
interface1:int3_1,
node2:n1,
interface2:int1_3).

Stage 2: Compute data plane

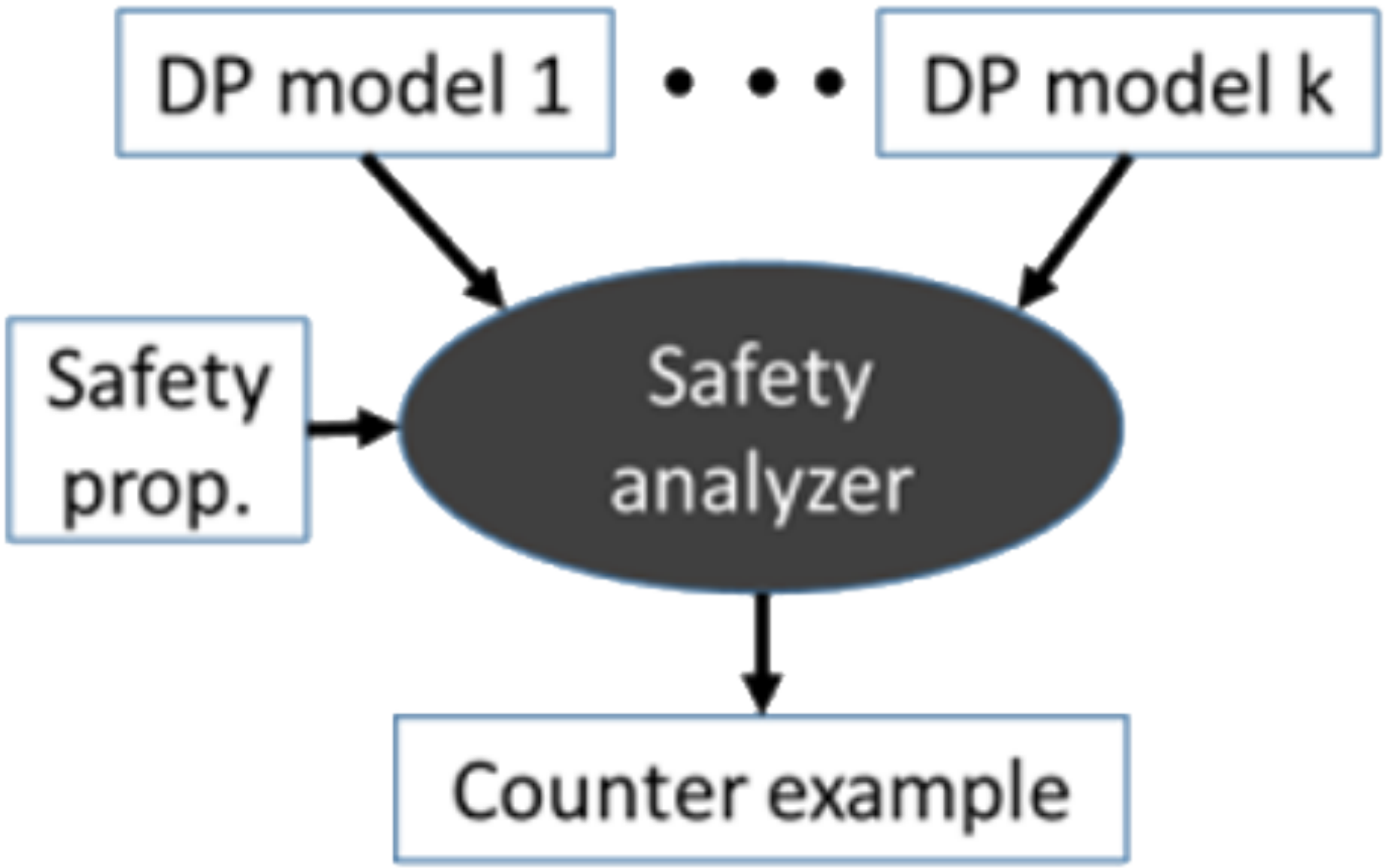



```
OspfExport(
        node=n2,
        network=10.0.0.0/24,
        cost=10,
        type=ospfE2).
```

```
InstalledRoute(route={
        node=n1,
        network=10.0.0.0/24,
        nextHop=n2
        administrativeCost=110,
        protocolCost=10,
        protocol=ospfE2}).
```

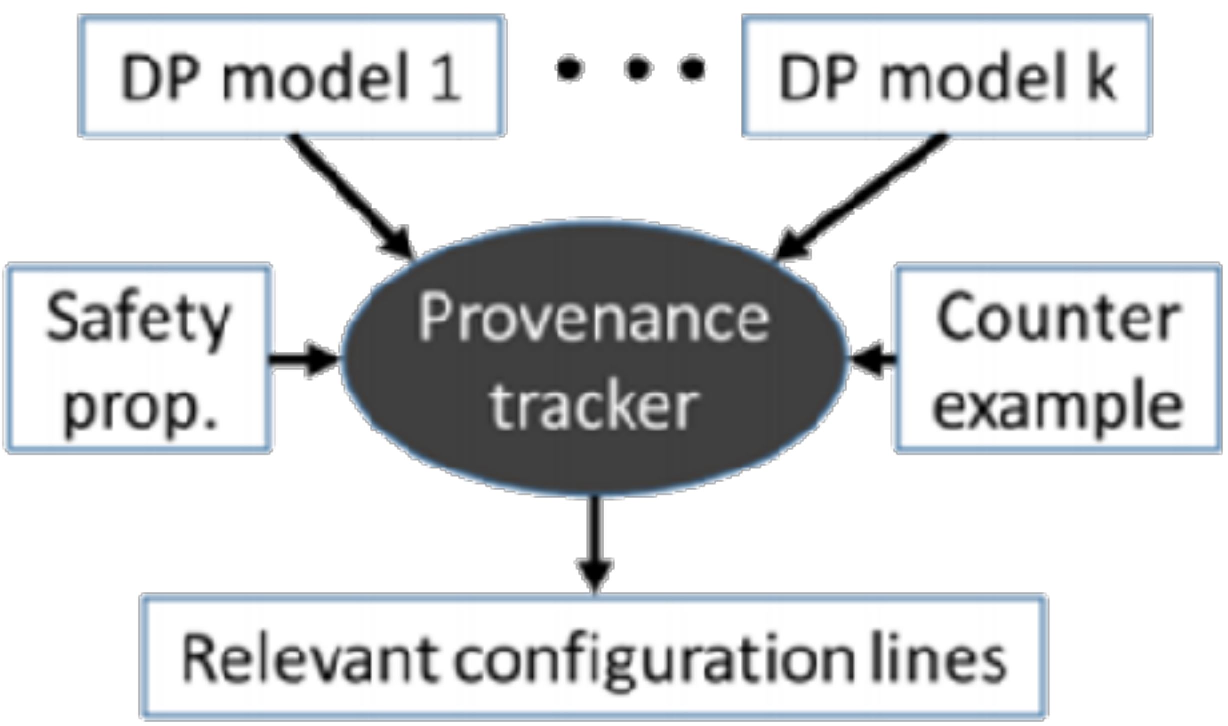
```
Fib(
        node=n1,
        network=10.0.0.0/24,
        egressInterface=int1_2).
```

Stage 3: Data plane analysis



Counterexample of multipath consistency
{
IngressNode=n1,
SrcIp=0.0.0.0,
DstIp=10.0.0.2,
IpProtocol=0
}

Stage 4: Report Provenance



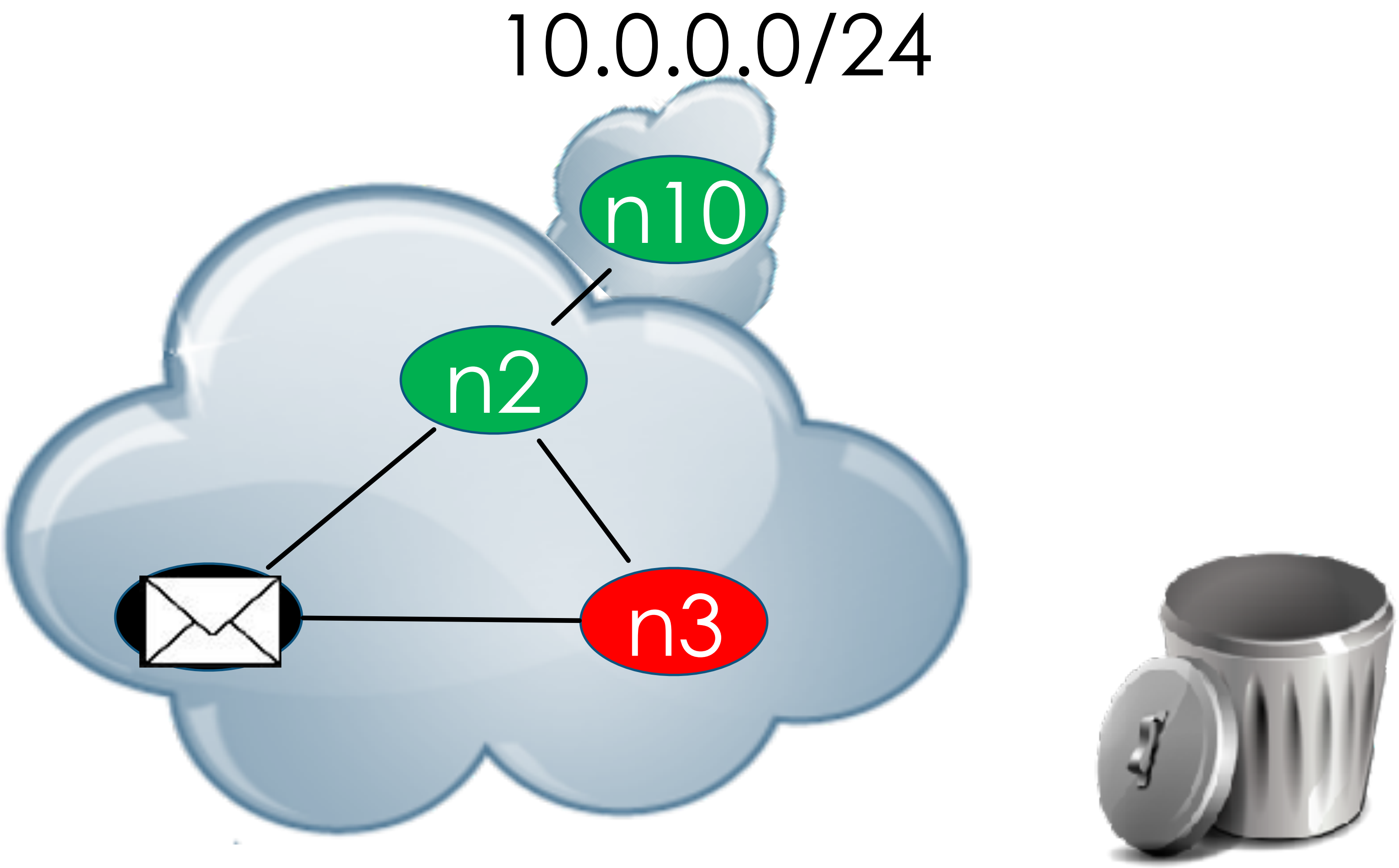
Counterexample packet traces

FlowPathHistory(
flow={ node=n1, … ,dstIp=10.0.0.2 },
1st hop:[n1:int1_2 → n2:int2_1]
2nd hop:[n2:int2_10 → n10:int10_2]
fate=accepted).

FlowPathHistory(
flow={ node=n1, … ,dstIp=10.0.0.2 },
1st hop:[n1:int1_3 → n3:int3_1]
fate=nullRouted by n3).

New Consistency Properties

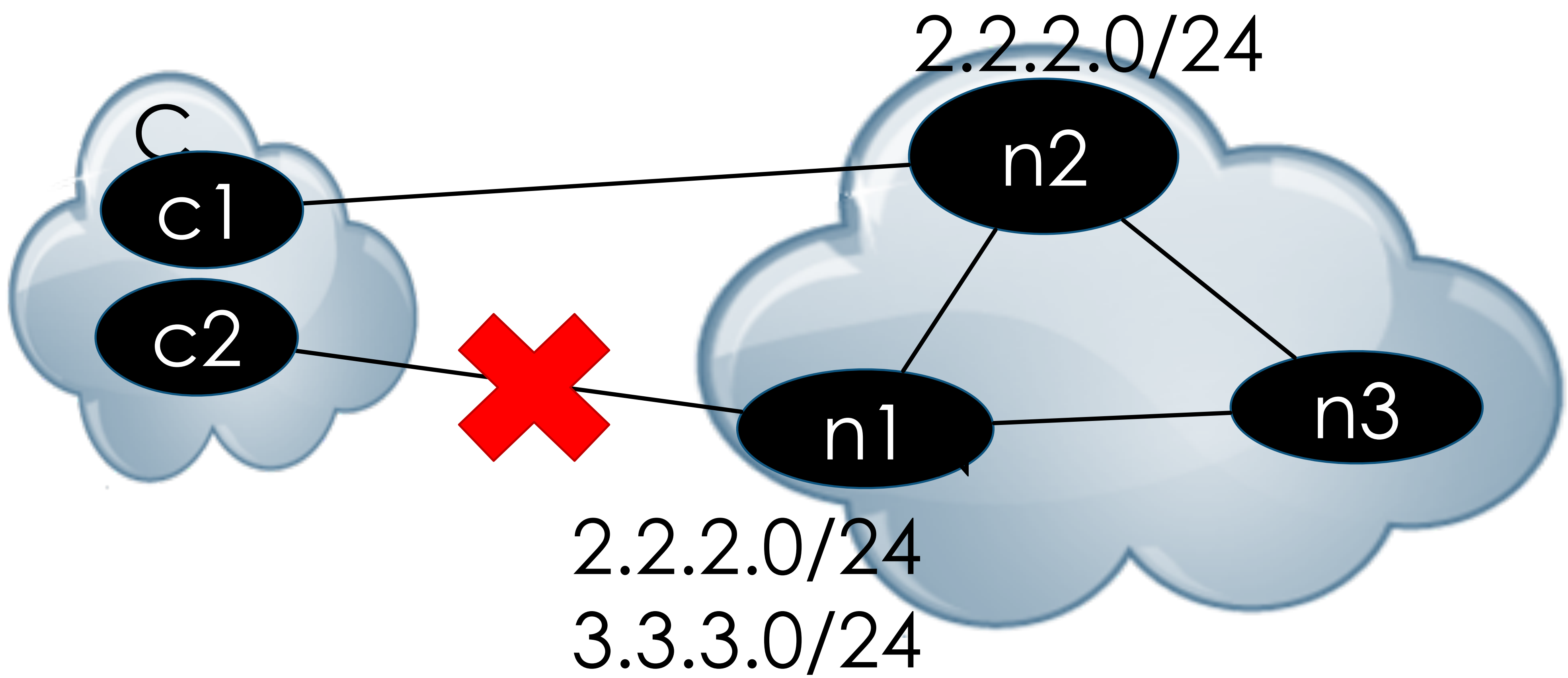
Multipath – disposition consistent on all paths



New Consistency Properties

Multipath – disposition consistent on all paths

Failure – reachability unaffected by failure

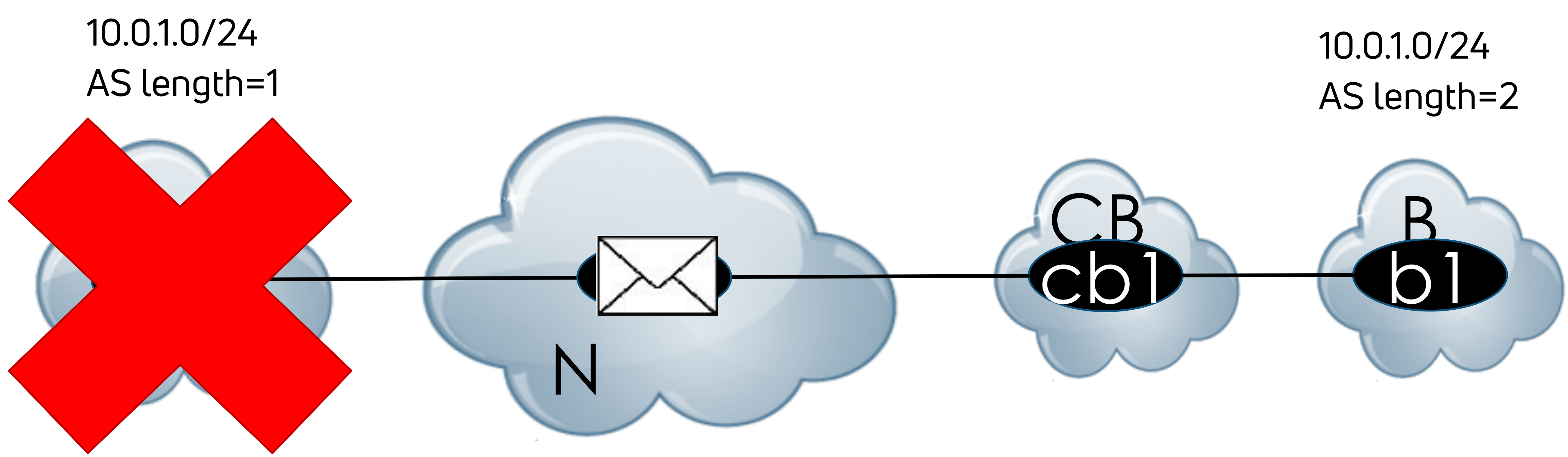


New Consistency Properties

Multipath – disposition consistent on all paths

Failure – reachability unaffected by failure

Destination – at most one customer per delegated address



Implementation

Support multiple configuration languages

- IOS, NX-OS, Juniper, Arista, …

Broad feature support

- Route redistribution, OSPF internal/external, BGP communities…

Unified, vendor-neutral intermediate representation

Evaluation

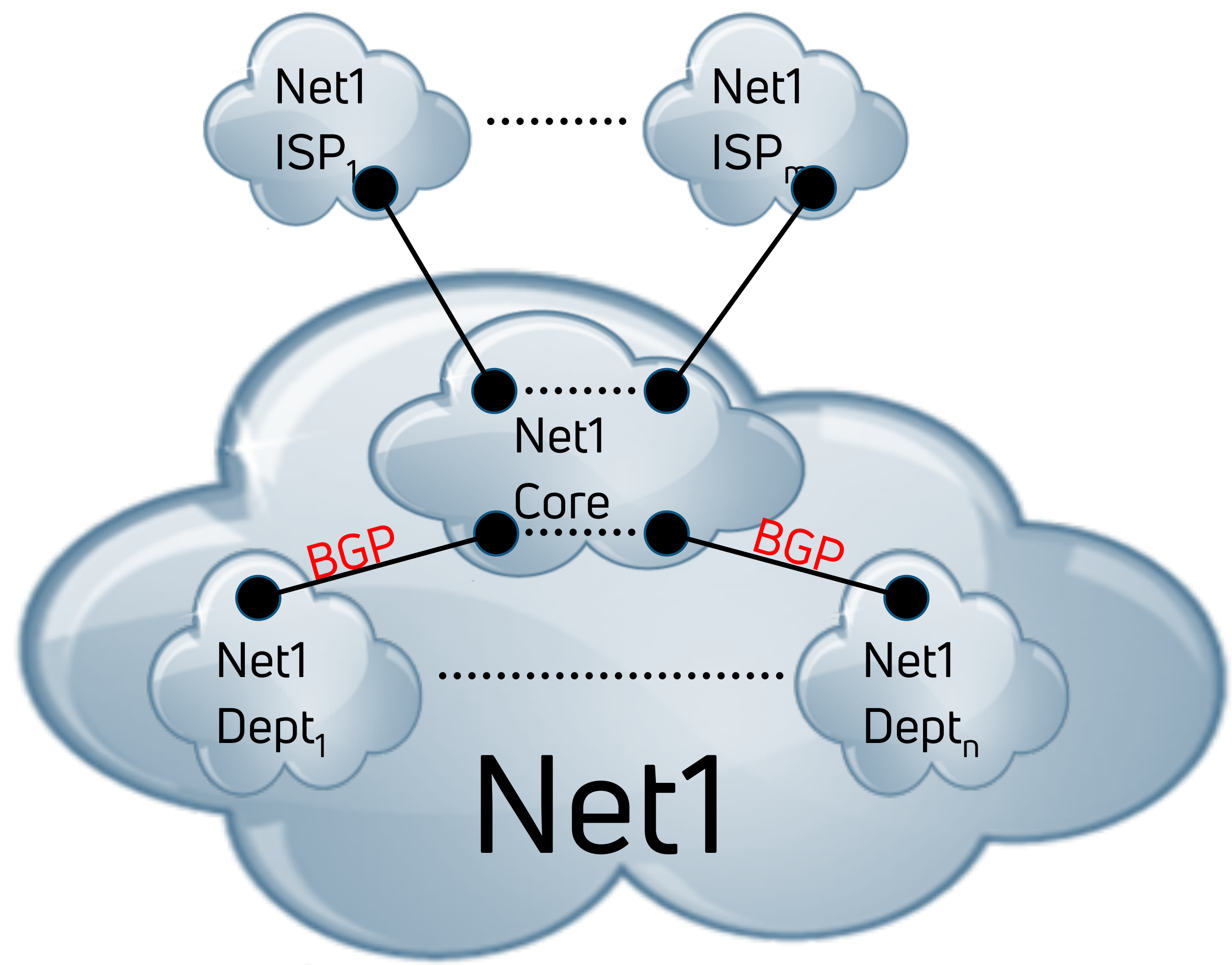
Two large university networks

Net1 – 21 core routers

- Federated network
- Each department is own AS
- Heavy use of BGP

Net2 – 17 core routers

- Centrally controlled
- Heavy use of VLANs
- Single AS
- BGP communication only with ISPs



Evaluation

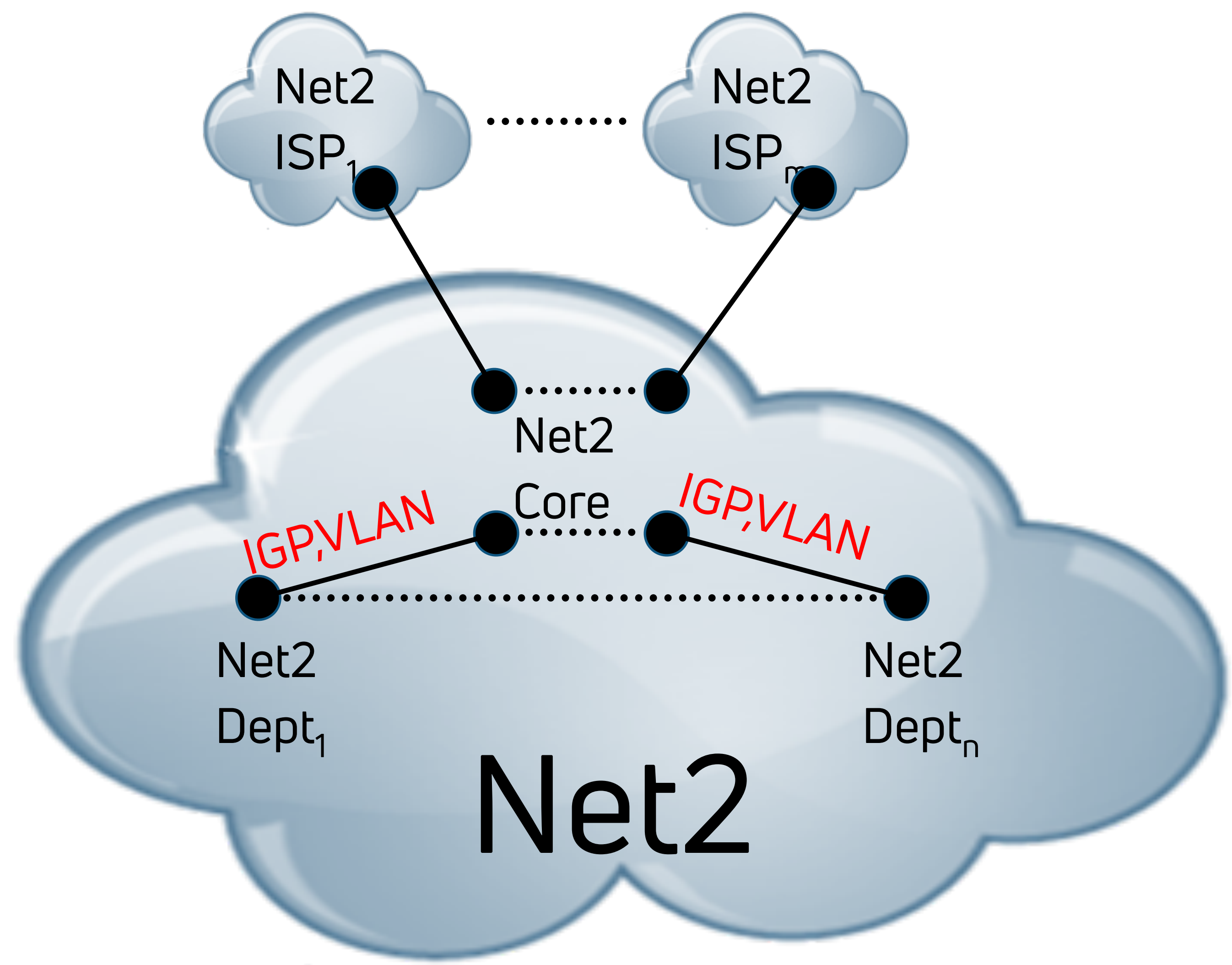
Two large university networks

Net1 – 21 core routers

- Federated network
- Each department is own AS
- Heavy use of BGP

Net2 – 17 core routers

- Centrally controlled
- Heavy use of VLANs
- Single AS
- BGP communication only with ISPs



Results

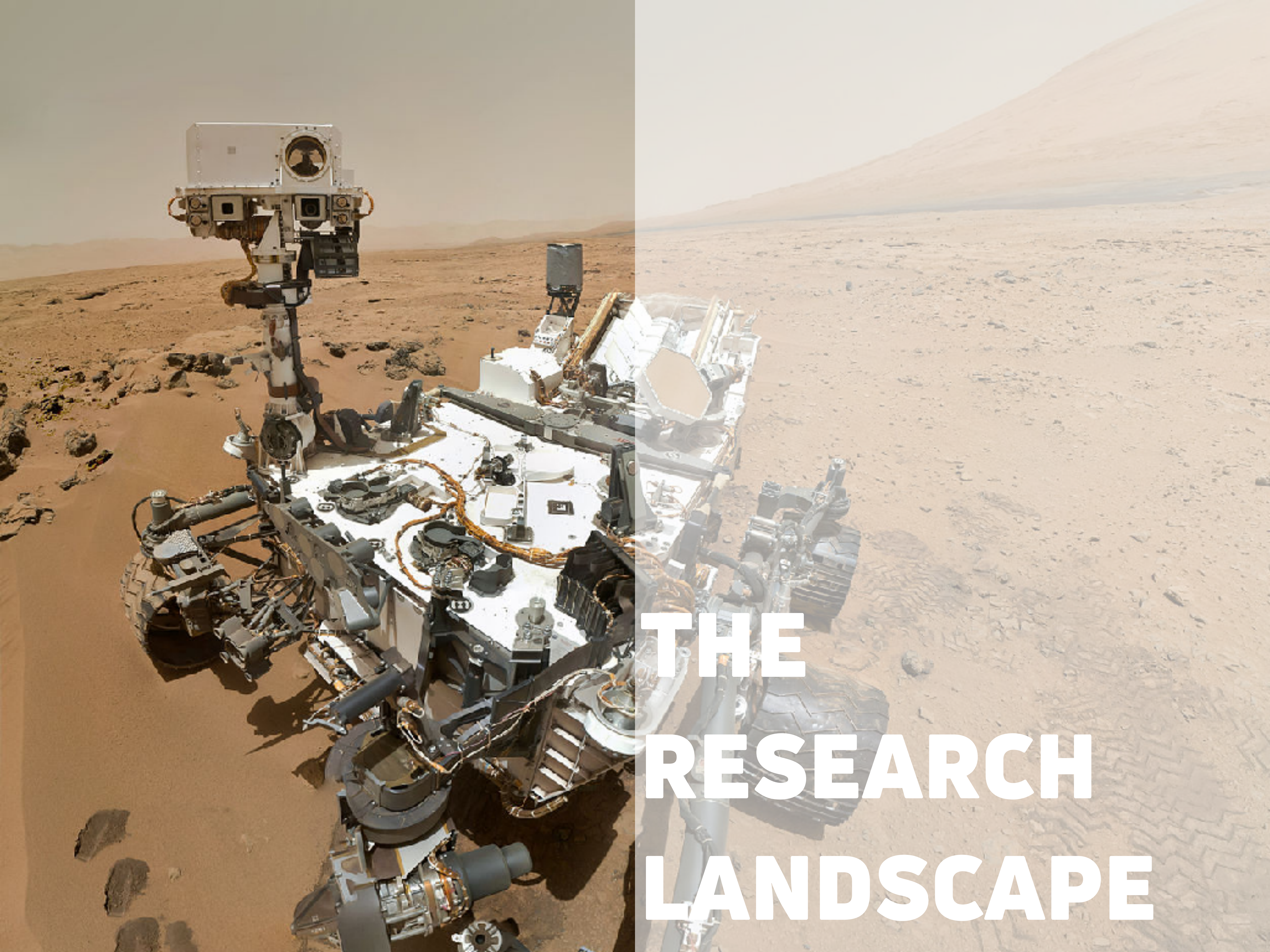
	Invariant	Total Violations	Violations Confirmed By Operators	Violations Fixed by Operators
Net1	Multipath	32(4)	32(4)	21(3)
	Failure	16(7)	3(2)	0(0)
	Destination	55(6)	55(6)	1(1)
Net2	Multipath	11(3)	11(3)	11(3)
	Failure	77(26)	18(7)	0(0)

Performance

	Data plane generation	Multipath consistency	Failure consistency
Net1	238 min	75 x (< 1.5 min)	199 x (~238 min)
Net2	37 min	17 x (< 1.5 min)	279 x (~37 min)

Comparing approaches

Configuration	• Enables "what-if" analysis • Trace root cause (in Batfish via LogicQL / LogicBlox)
Control software	
Data plane state	• Provides basis for higher-level analysis • Accuracy based on actual observed data plane
Packet processing	


THE
RESEARCH
LANDSCAPE

Data plane verification

Static

- On static reachability in IP networks [Xie, Zhan, Maltz, Zhang, Greenberg, Hjalmtysson, Rexford, INFOCOM '05]
  - Essentially early form of data plane verification
  - Computed reachable sets with IP forwarding rules
- FlowChecker [Al-Shaer, Al-Haj, SafeConfig '10]
- Anteater [Mai, Khurshid, Agarwal, Caesar, G., King, SIGCOMM'11]
- Header Space Analysis [Kazemian, Varghese, and McKeown, NSDI '12]
- Network-Optimized Datalog (NoD) [Lopes, Bjørner, Godefroid, Jayaraman, Varghese, NSDI 2015]

Real time (incremental)

- VeriFlow [Khurshid, Zou, Zhou, Caesar, G., HotSDN'12, NSDI'13]
- NetPlumber [Kazemian, Chang, Zeng, Varghese, McKeown, Whyte, NSDI '13]
- CCG [Zhou, Jin, Croft, Caesar, G., NSDI'15]

Data plane verification (cont'd)

Optimizations

- Libra: Divide and Conquer to Verify Forwarding Tables in Huge Networks [Zeng, Zhang, Ye, Google, Jeyakumar, Ju, Liu, McKeown, Vahdat, NSDI'14]
- Atomic Predicates [Yang, Lam, ToN'16]
- ddNF [Bjorner, Juniwal, Mahajan, Seshia, Varghese, HVC'16]

Configuration verification

Configuration verification

- RCC (Detecting BGP config faults w/static analysis) [Feamster & Balakrishnan, USENIX '05]
- ConfigAssure [Narain et al, '08]
- ConfigChecker [Al-Shaer, Marrero, El-Atawy, ICNP '09]
- Batfish [Fogel, Fung, Pedrosa, Walraed-Sullivan, Govindan, Mahajan, Millstein, NSDI'15]
- Bagpipe [Weitz, Woos, Torlak, Ernst, Krishnamurthy, Tatlock, NetPL'16 & OOPSLA'16]

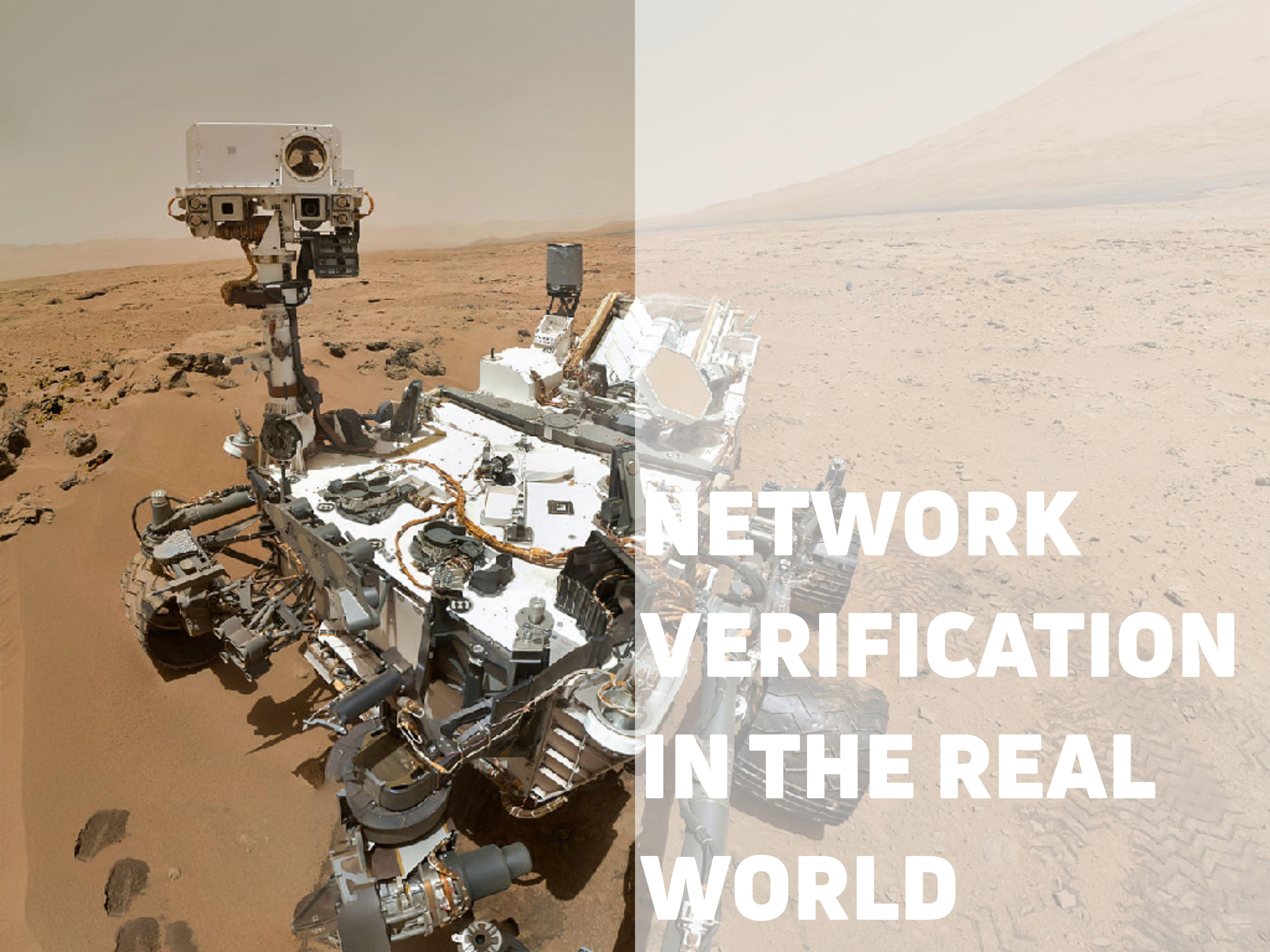
Richer verification

Richer data plane models

- Software Dataplane Verification [Dobrescu, Argyraki, NSDI'14]
- SymNet [Stoenescu, Popovici, Negreanu, Raiciu, SIGCOMM'16]
- Mutable datapaths [Panda, Lahav, Argyraki, Sagiv, Shenker, NSDI'17]

Verifiable controllers & control languages

- NICE [Canini, Venzano, Perešíni, Kostić, Rexford, NSDI'12]
- NetKAT [Anderson, Foster, Guha, Jeannin, Kozen, Schlesinger, Walker, POPL'14]
- Kinetic: Verifiable Dynamic Network Control [Kim, Gupta, Shahbaz, Reich, Feamster, Clark, NSDI'15]



NETWORK VERIFICATION IN THE REAL WORLD

Industry efforts

Three startups pursuing general-purpose network verification for enterprises

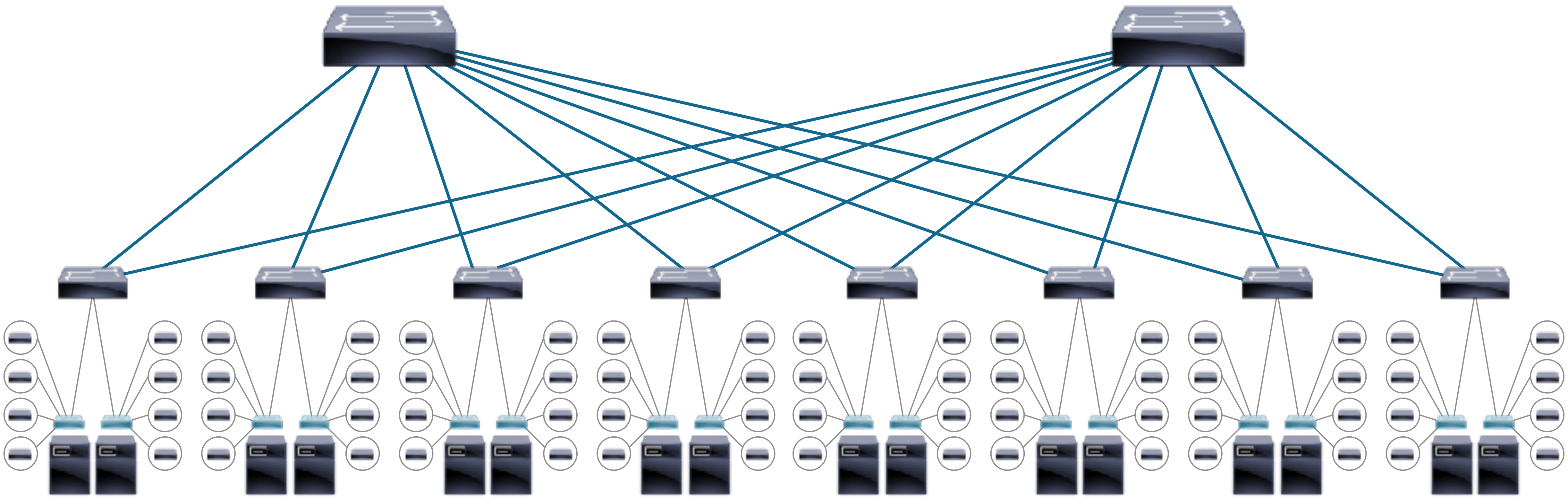
Special purpose efforts

- Hyperscale clouds
- Major network device manufacturer

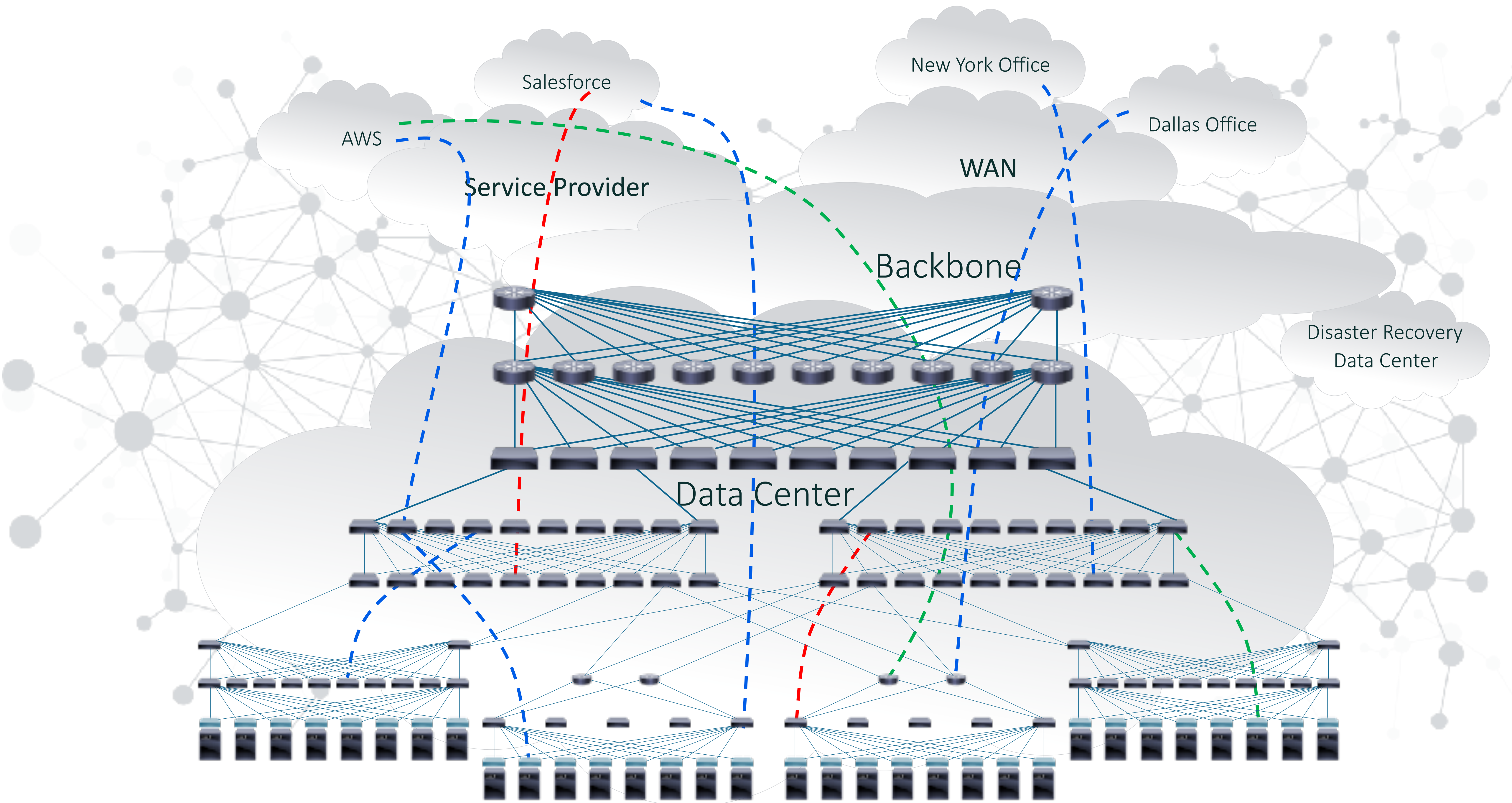
Gartner grouping verification in “intent-based networking” category

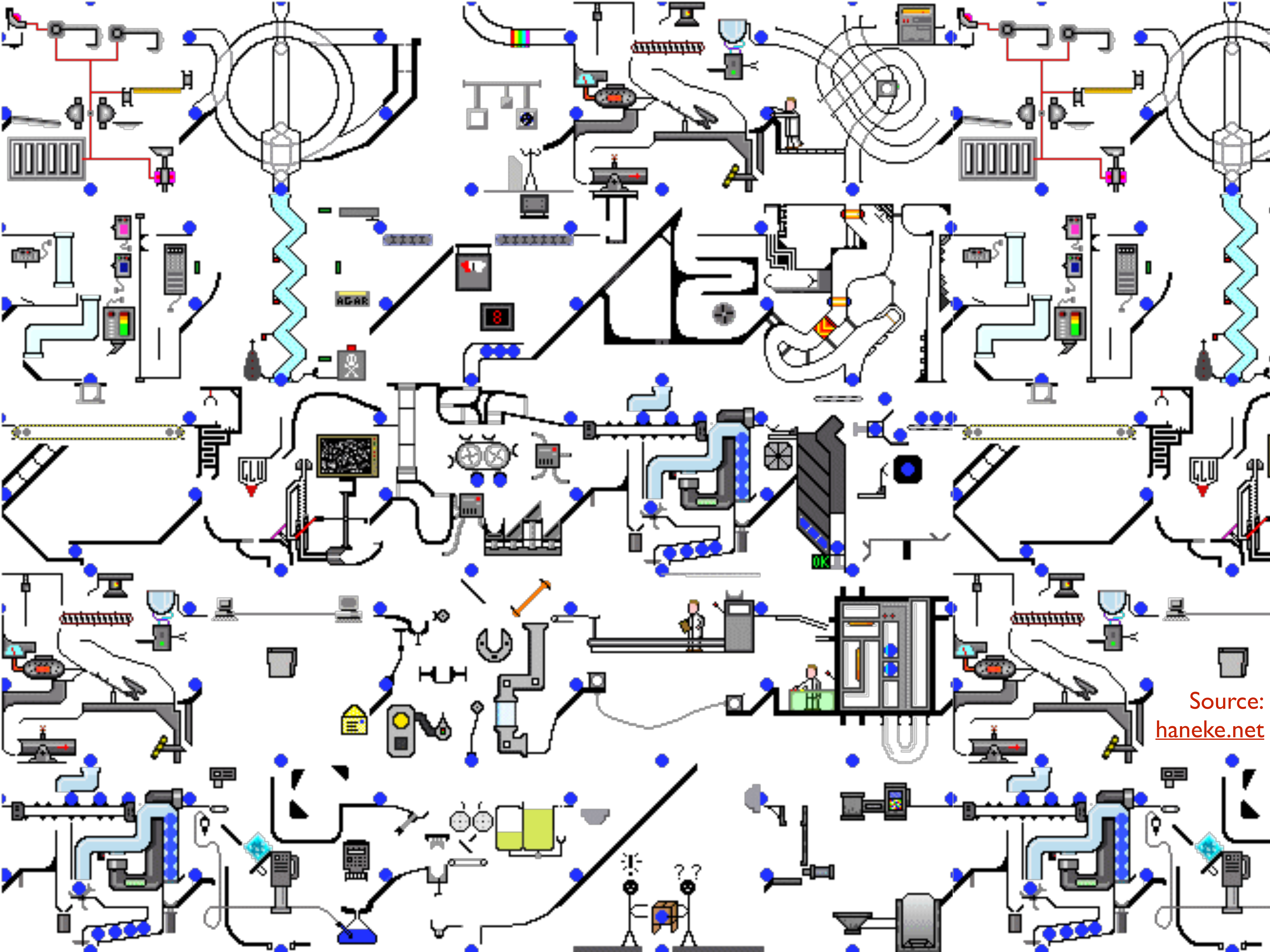
What have we learned?

1. The Need is Real



1. The Need is Real





Source: haneke.net

1. The Need is Real

Network Complexity

59% say growth in complexity has led to more frequent outages [Dimensional Research]

Change

22,000 changes/mo. at DISA [S. Zabel, 2016]

Manual Processes

69% use manual checks (most common technique) [Dimensional Research]

2. How is it actually useful?

Network Segmentation

Availability & Resilience

Continuous Compliance

Incident Response

3. Extracting the abstraction: not easy

Software verification

```
#include <stdio.h>

int main(int argc, char** argv) {
    if (argv >= 2) {
        printf("Hello world, %s!",
            argv[1]);
    }
    return 0;
}
```

- Given program as input
- Assume formal specification of programming language

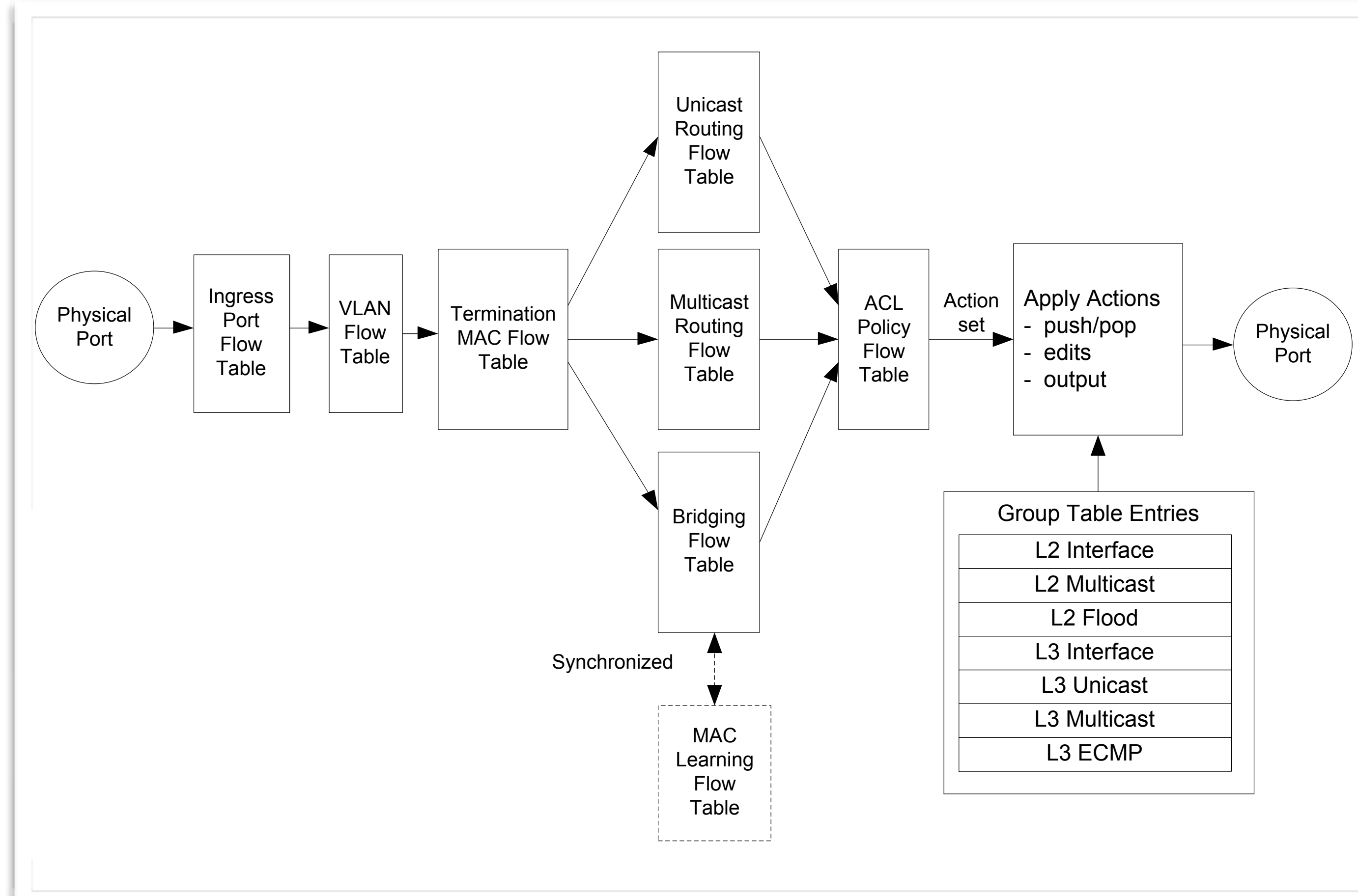
Data plane verification

- No universal API to extract state (LCD: SSH + CLI "show" commands")
- No formal spec of how that state relates to functionality
- Vendor-specific behaviors

3. Extracting the abstraction: not easy

Data plane verification

- No universal API to extract state (LCD: SSH + CLI "show" commands")
- No formal spec of how that state relates to functionality
- Vendor-specific behaviors



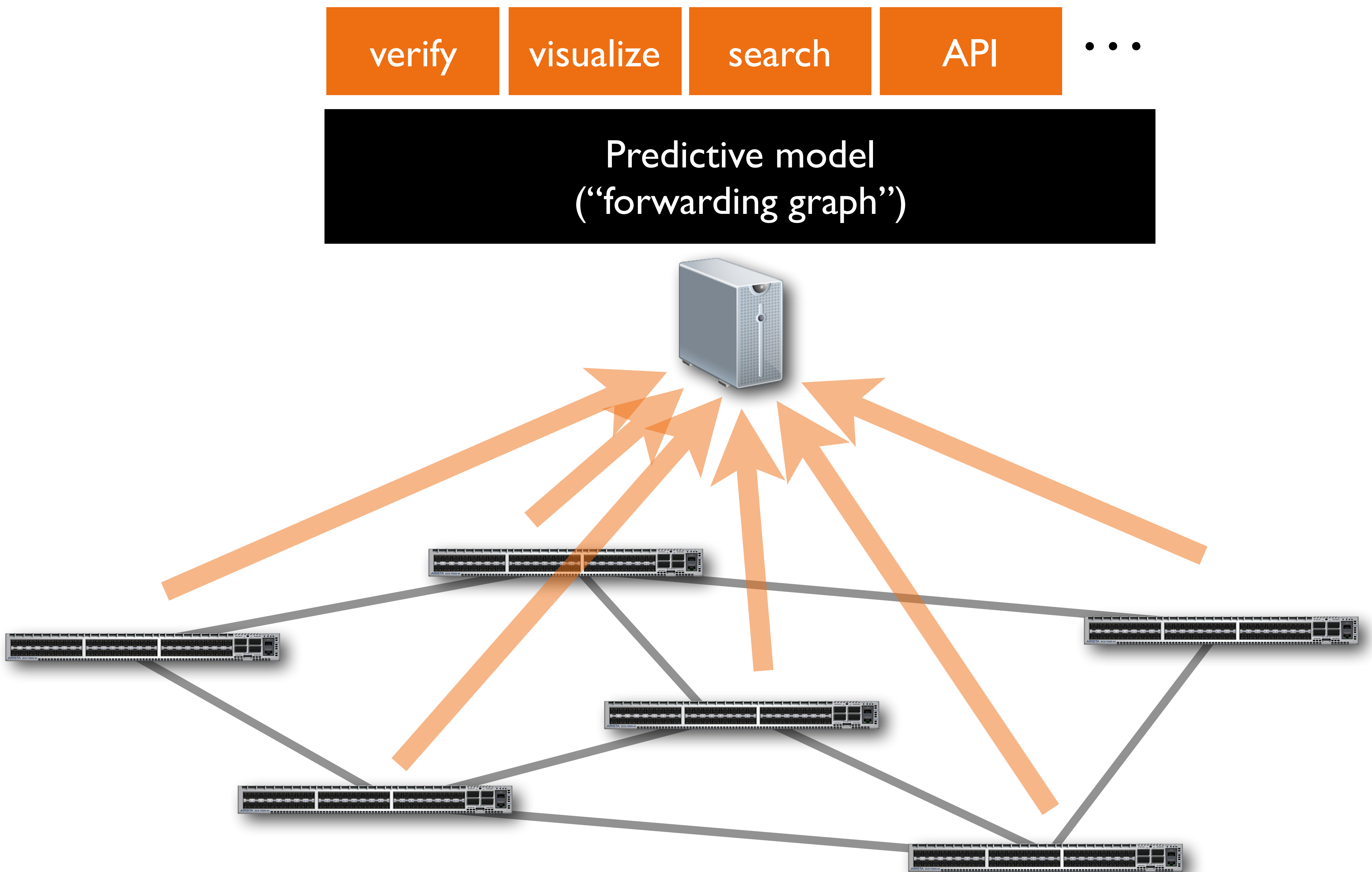
Broadcom's OF-DPA 1.0 Abstract Switch

https://www.ietf.org/proceedings/90/slides/slides-90-sdnrg-3.pdf

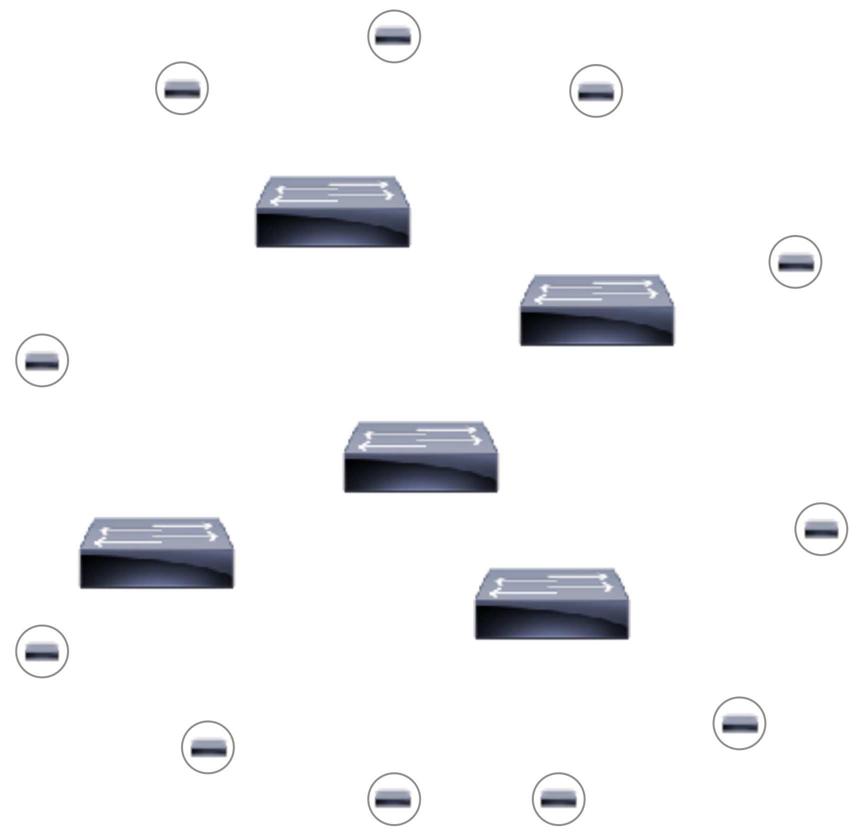
- Some hope: Vendor-specific APIs, OpenConfig

4. Model / Verifier separation works

4. Model / Verifier separation works



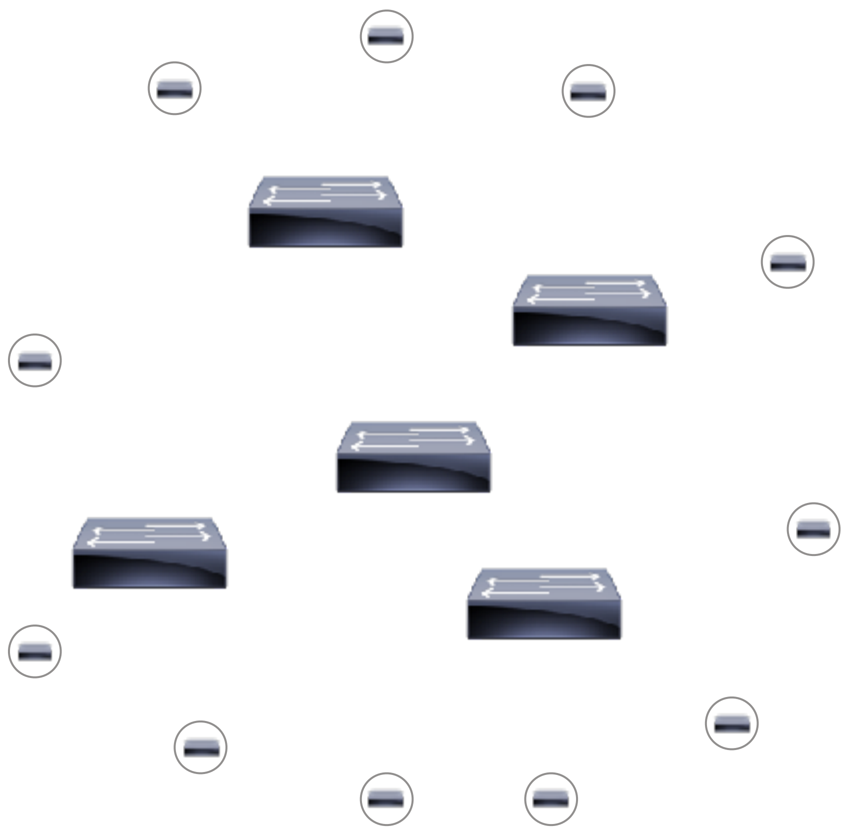
5. We need a shift in thought

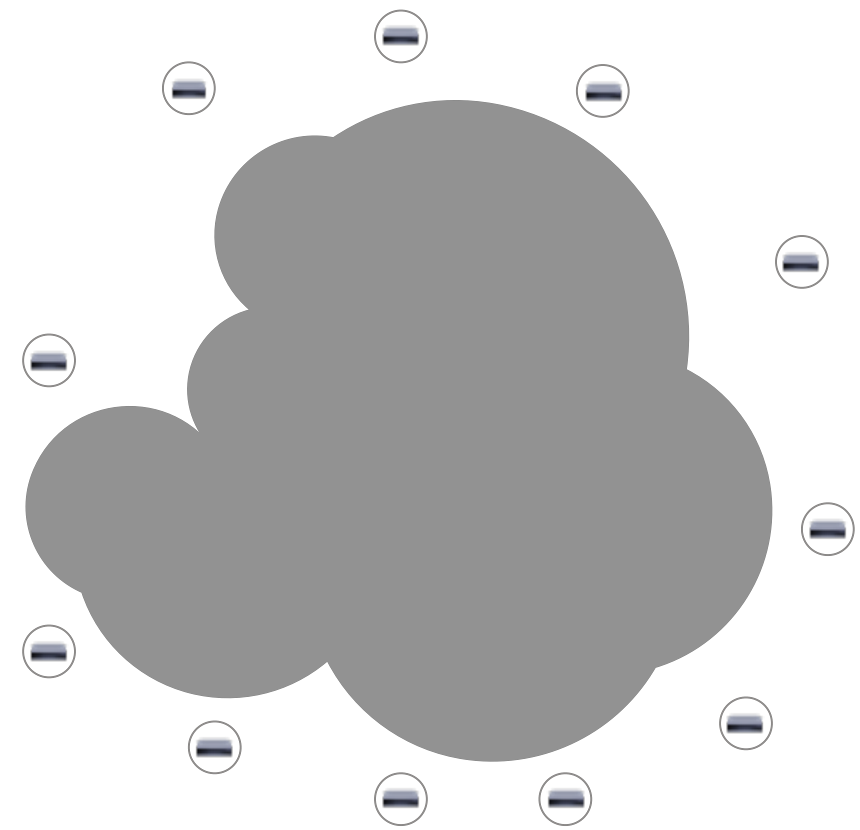


Network as individual devices

Individual config knobs

5. We need a shift in thought





Network as individual devices → Network as one system

Individual config knobs → End-to-end intent


THANK YOU!