

Network Verification: From Algorithms to Deployment



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

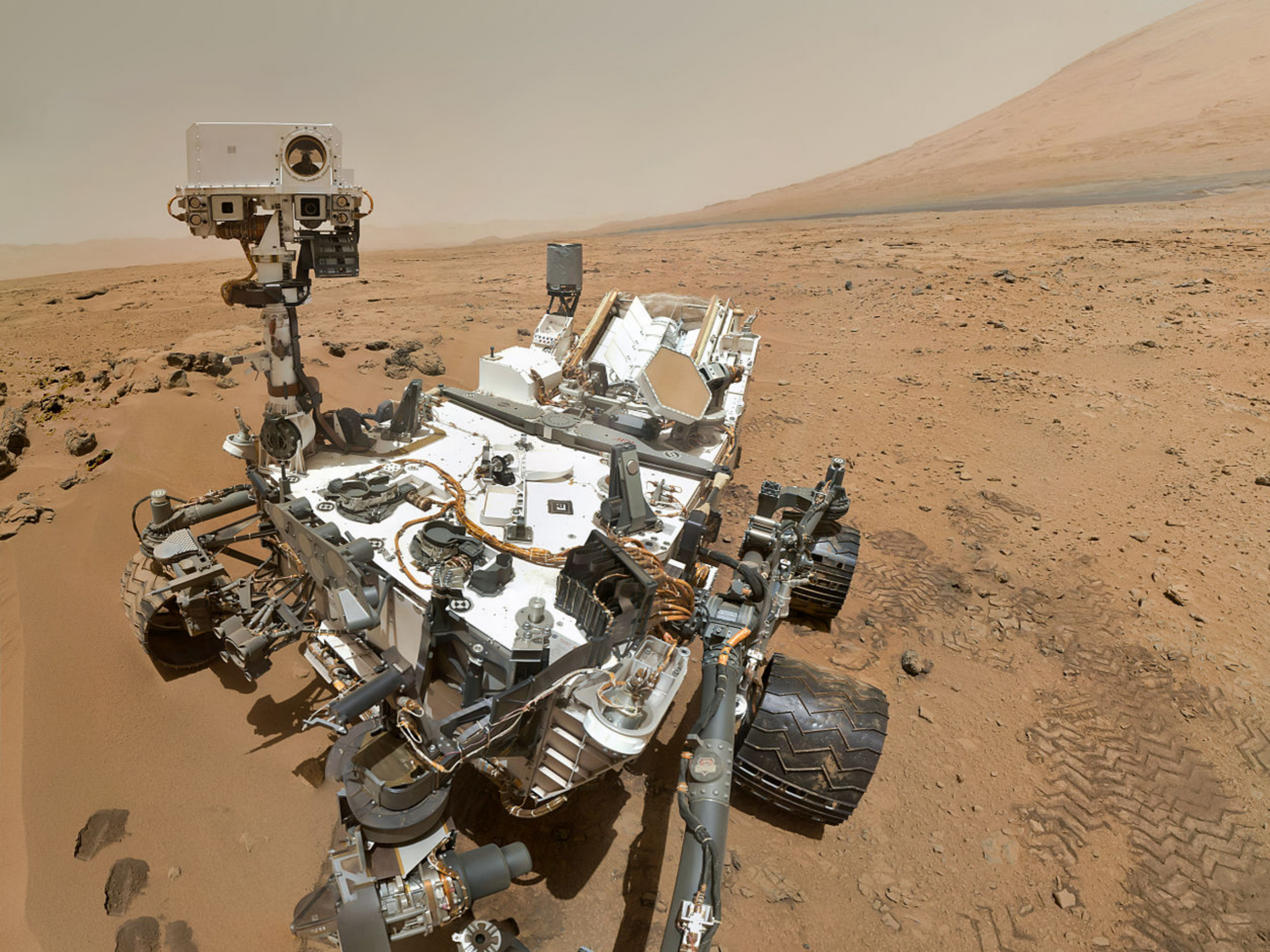


Santhosh Prabhu
PhD Candidate, UIUC
Engineer, Core Team, Veriflow

Tutorial at ASE 2017
November 3, 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

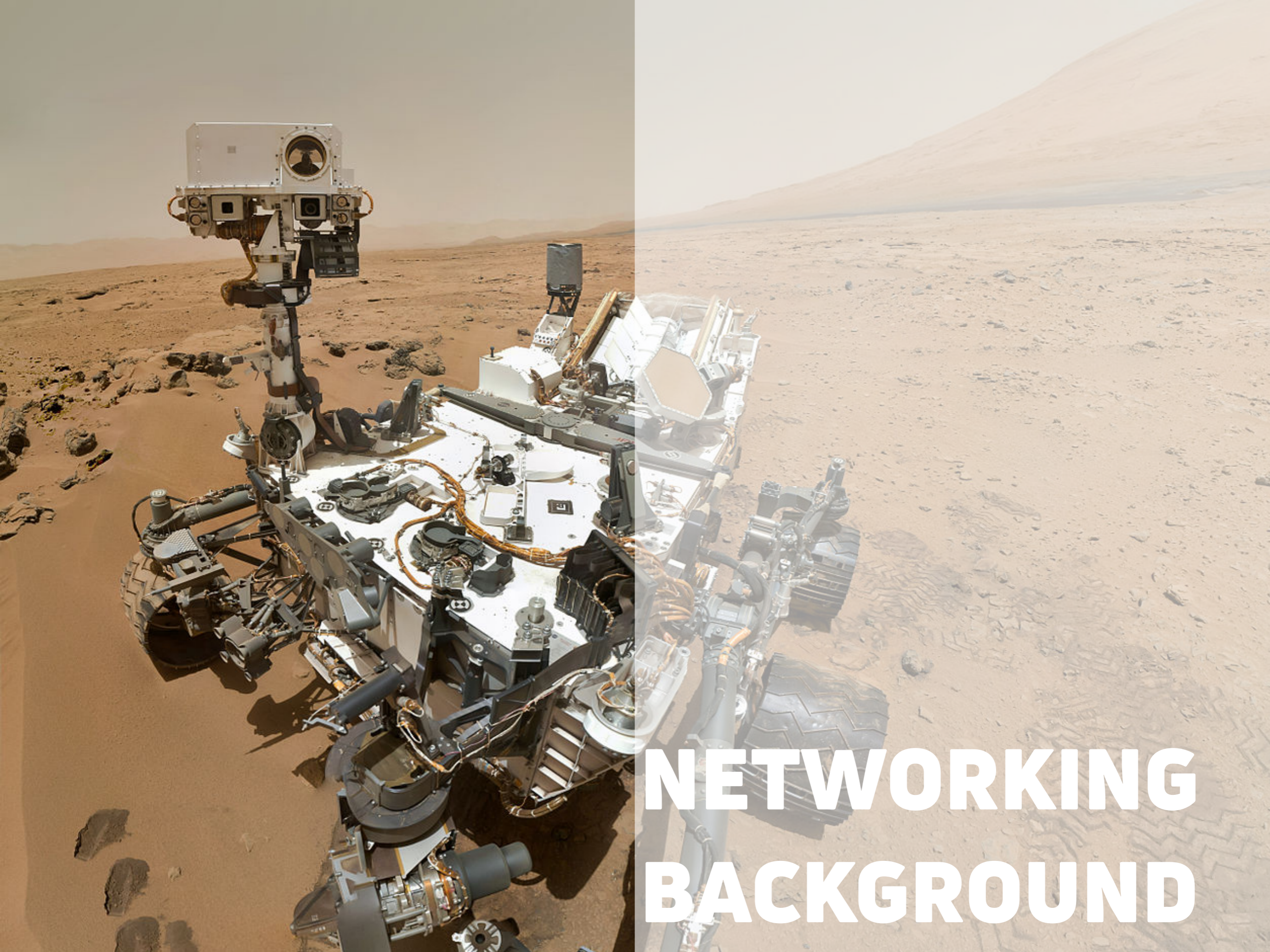
Configuration verification

- Simulation- and Emulation-based
- Formal Analysis

Scaling network verification

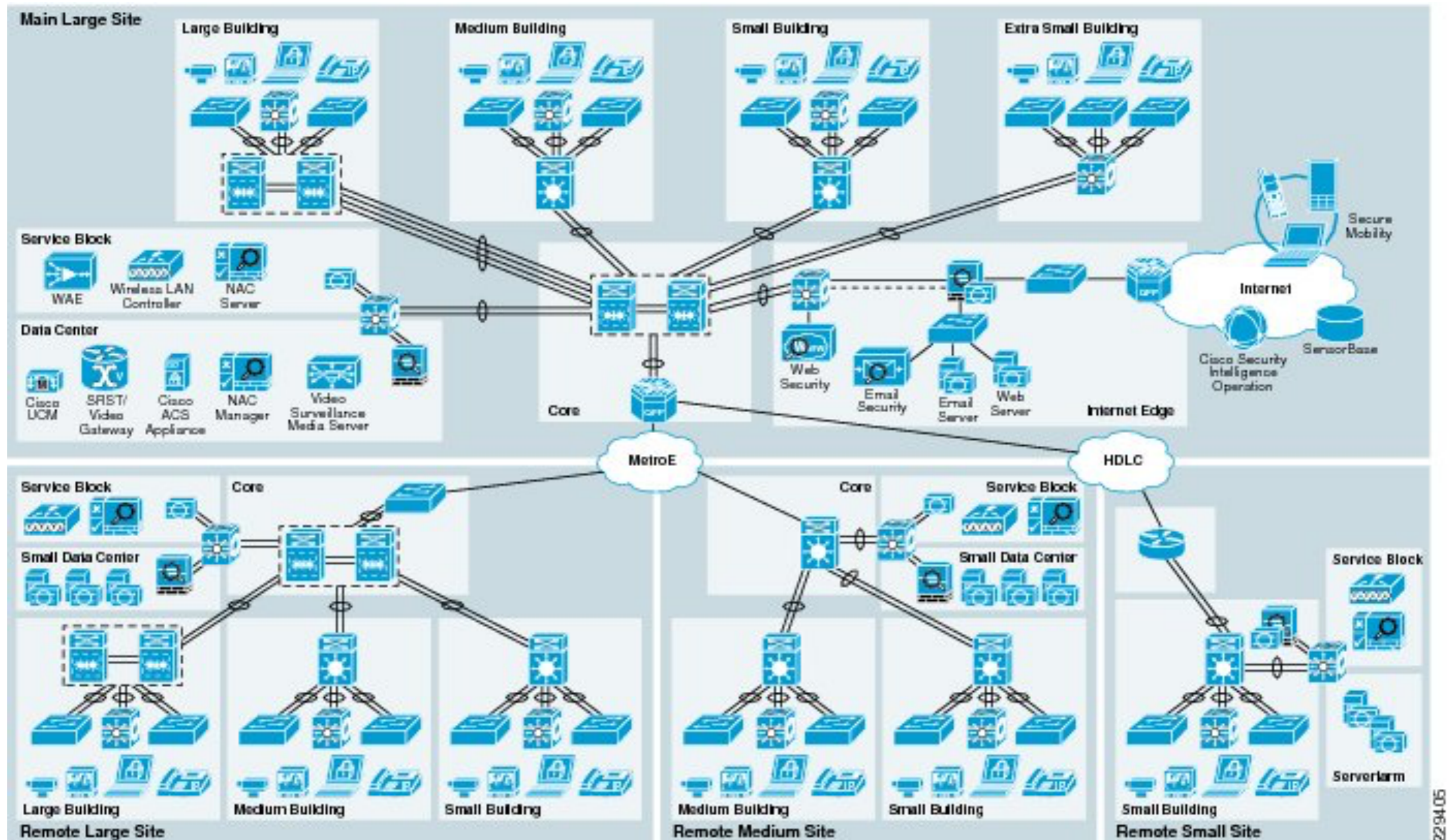
Research landscape & directions

Network verification in the real world

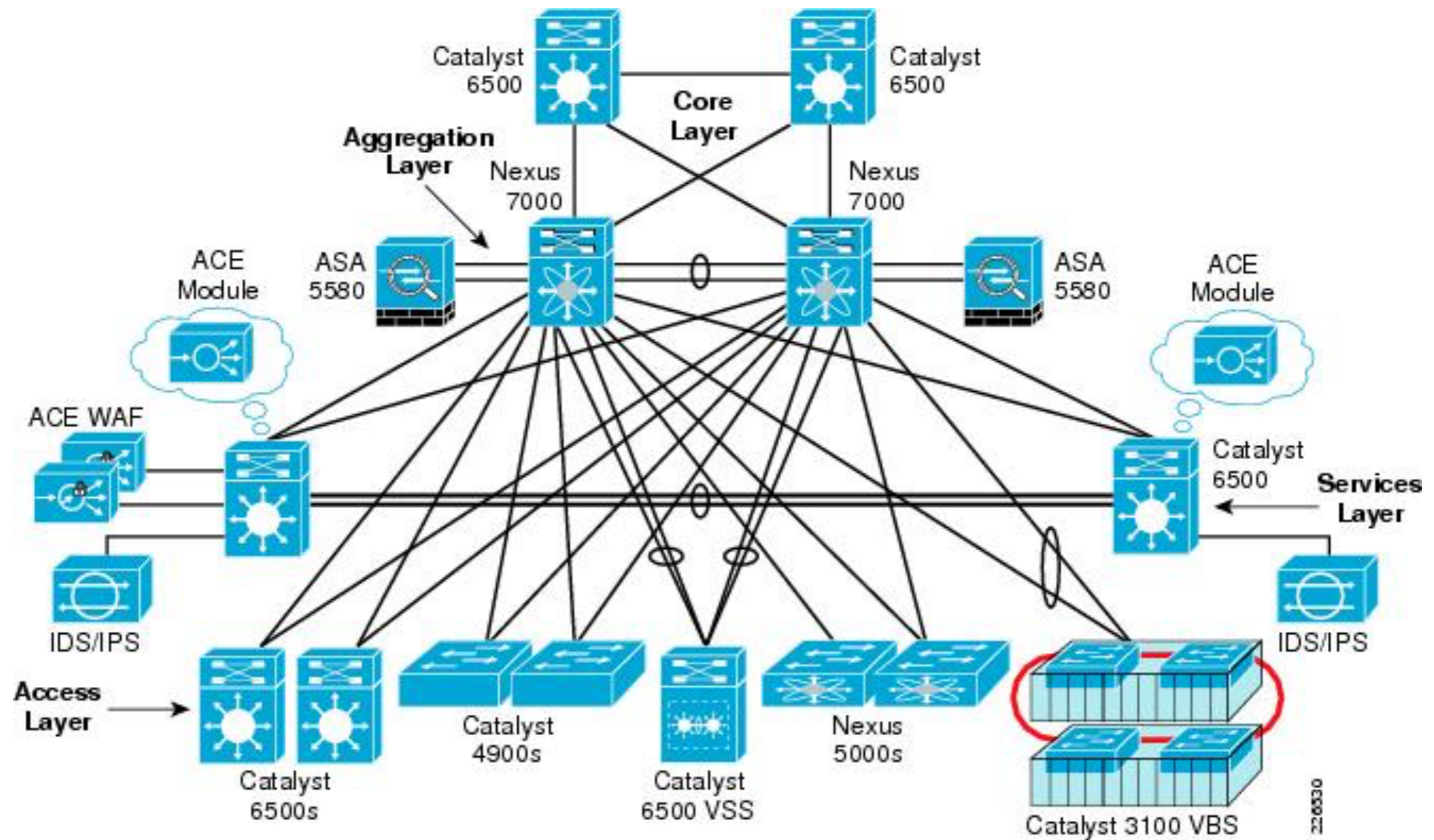


**NETWORKING
BACKGROUND**

Inside a typical enterprise network



Inside a typical enterprise data center



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

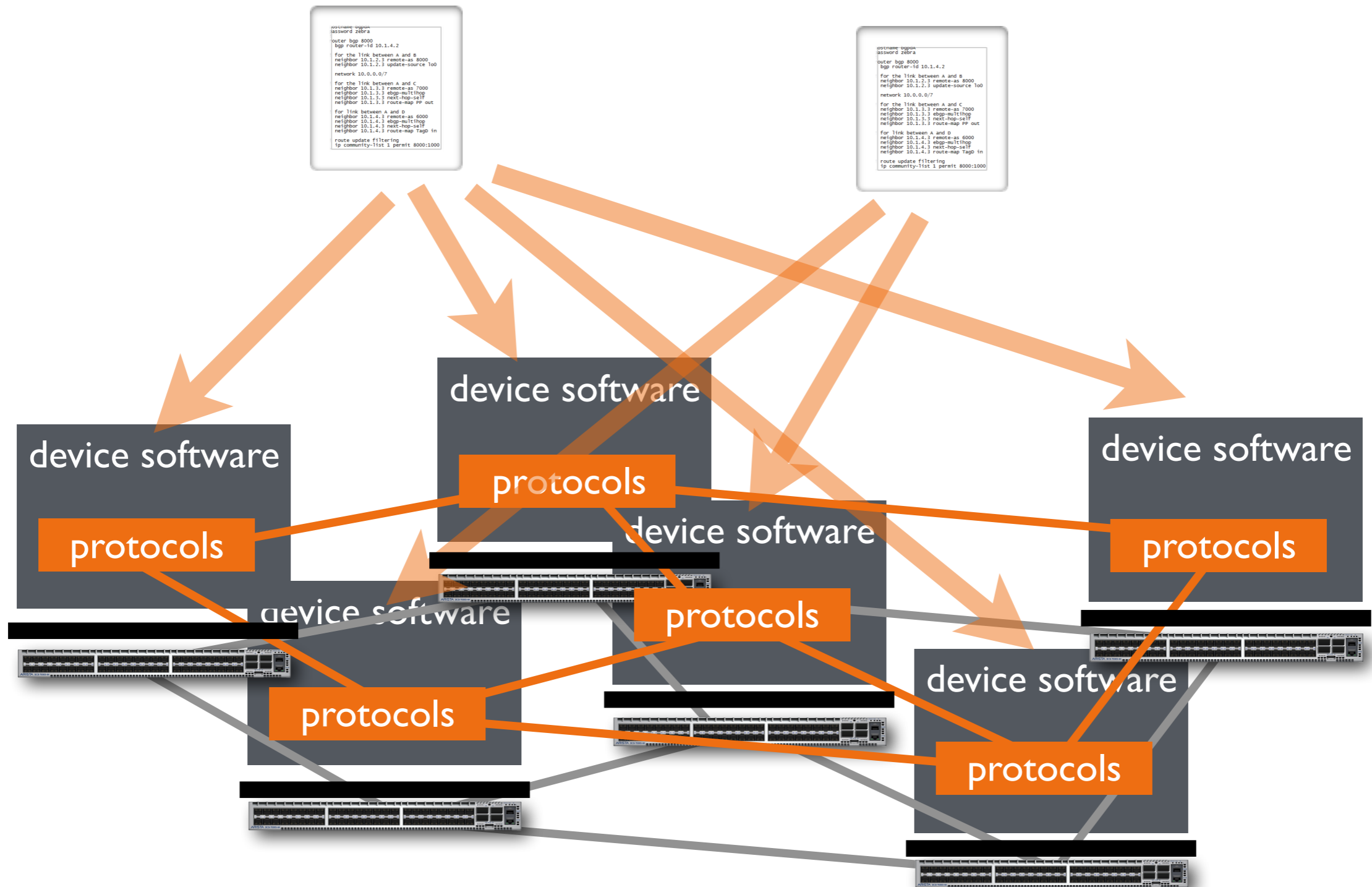
```
version 12.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname PrimaryR1
!
boot-start-marker
boot-end-marker
!
!
no aaa new-model
!
!
ip cef
!
interface Loopback100
no ip address
!
interface GigabitEthernet0/1
description LAN port
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
!
```

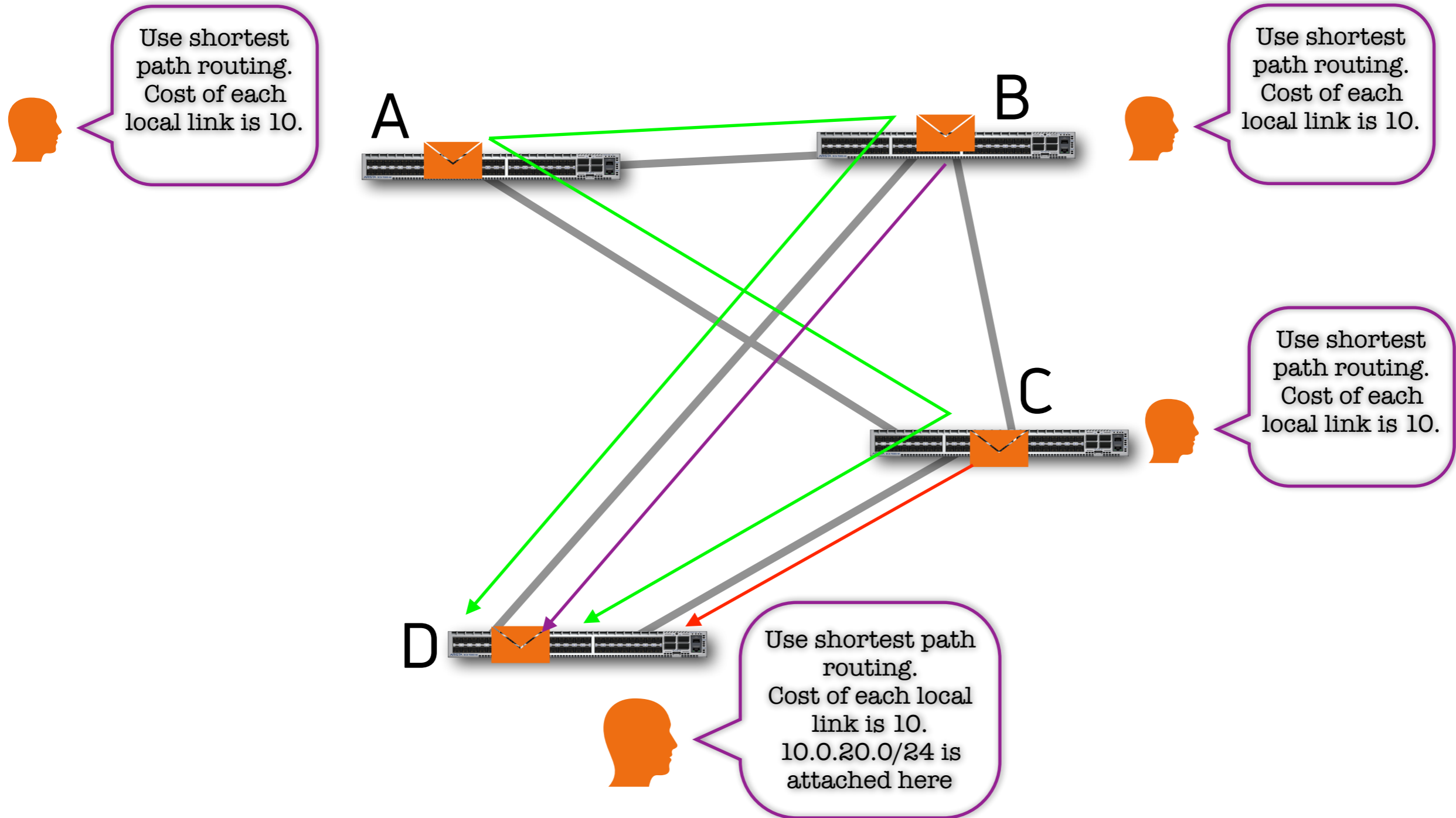
```
router bgp 16XX
no synchronization
bgp log-neighbor-changes
network 64.X.X.0 mask 255.255.255.224
network 64.X.X.2
aggregate-address 64.X.X.0 255.255.255.0 summary-only
neighbor 64.X.X.2 remote-as 16XX
neighbor 64.X.X.2 next-hop-self
neighbor 65.X.1X.253 remote-as 2828
neighbor 65.X.X.253 route-map 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>

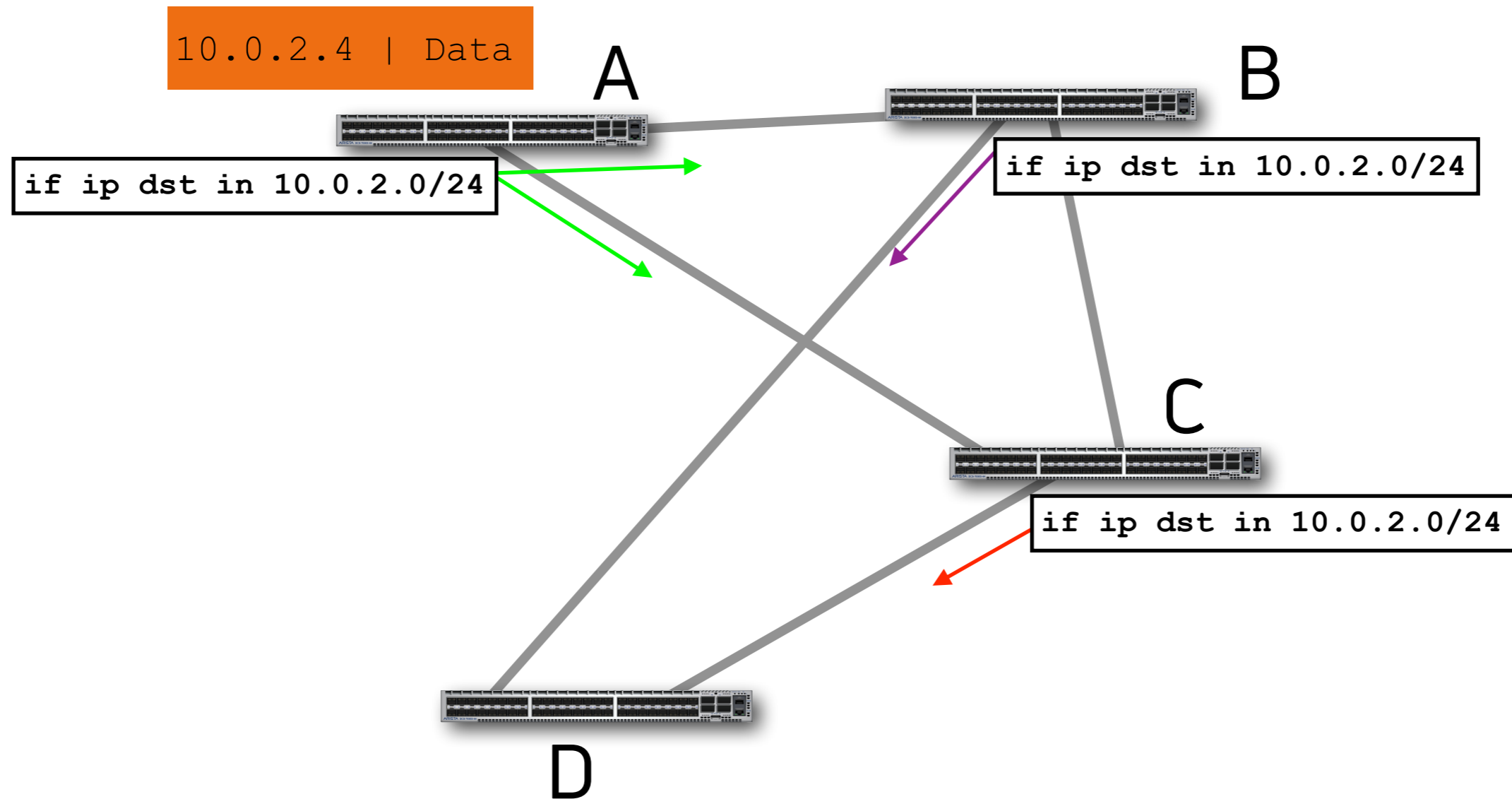
Distributed route computation



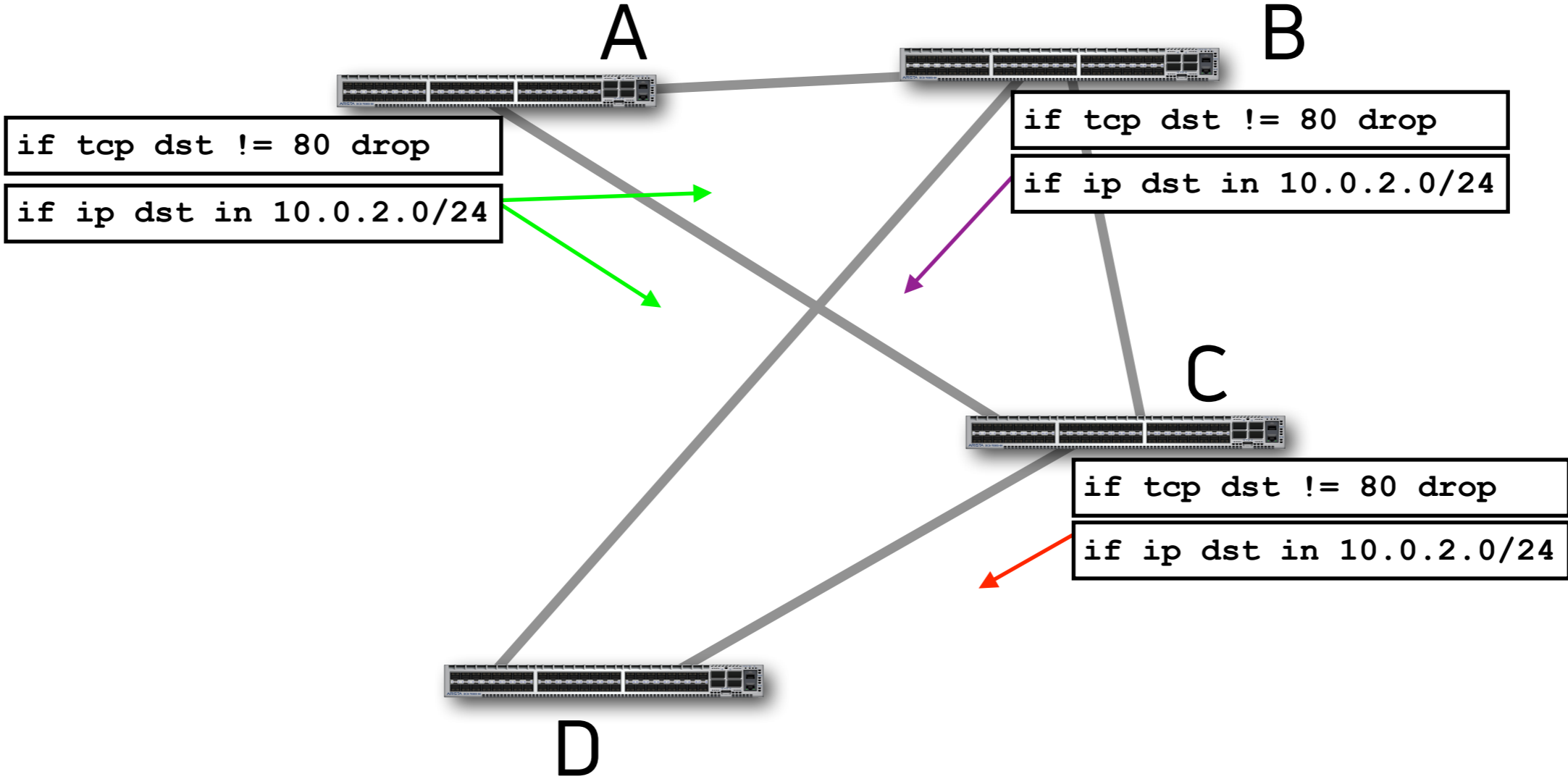
Routing Example - Shortest Paths



Routing Example - Forwarding

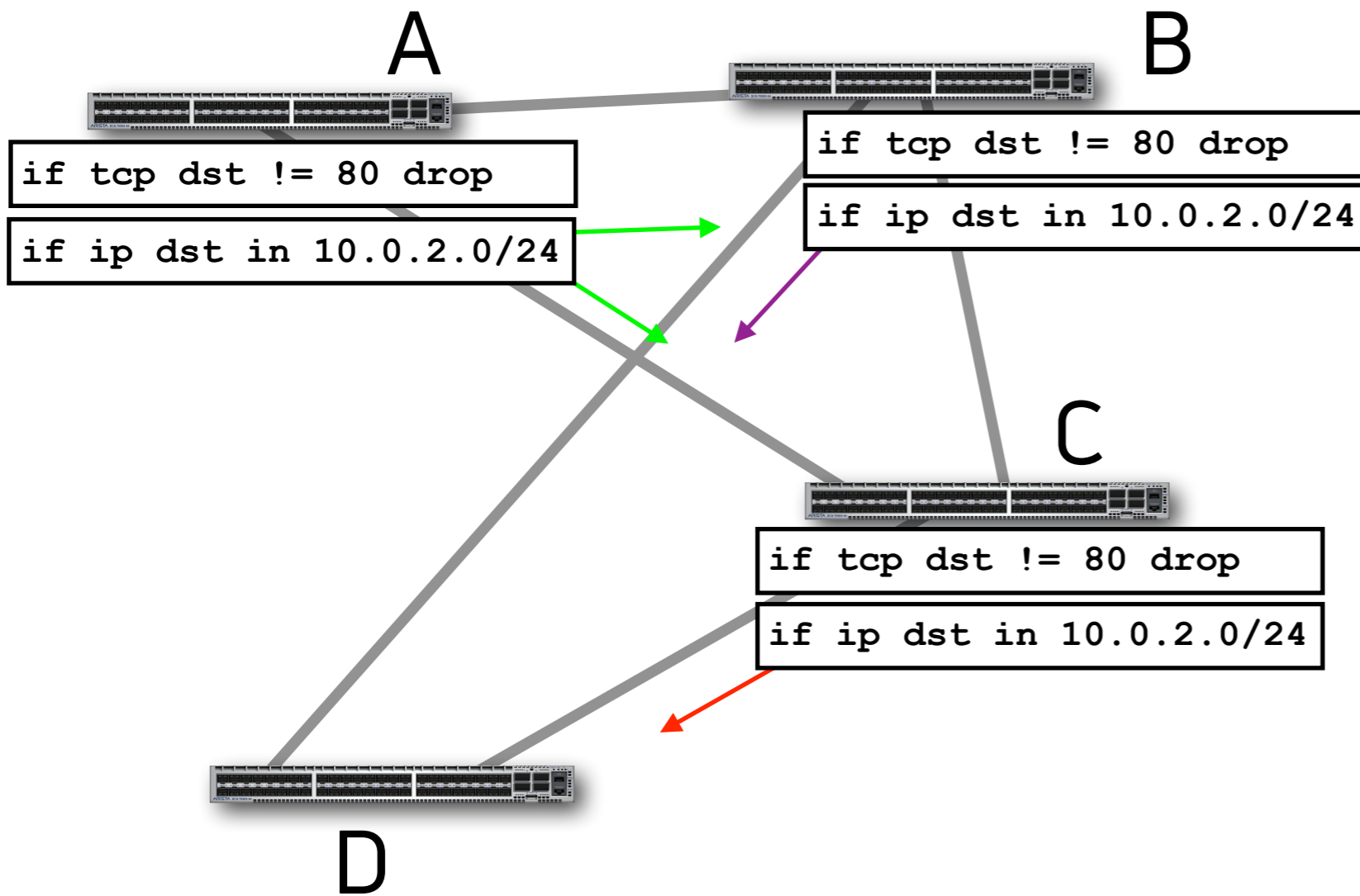


Access Control



Allow only HTTP

Network Correctness (a.k.a Intent)



Intent Checklist

- All packets should follow the shortest paths
- There should not be any loops
- There should be multiple paths between A and D
- HTTP Packets in the subnet 10.0.2.0/24 should reach D
- External networks attached to A should be isolated from D except via HTTP

Networks are complex

89%

of operators never sure
that config changes are
bug-free

82%

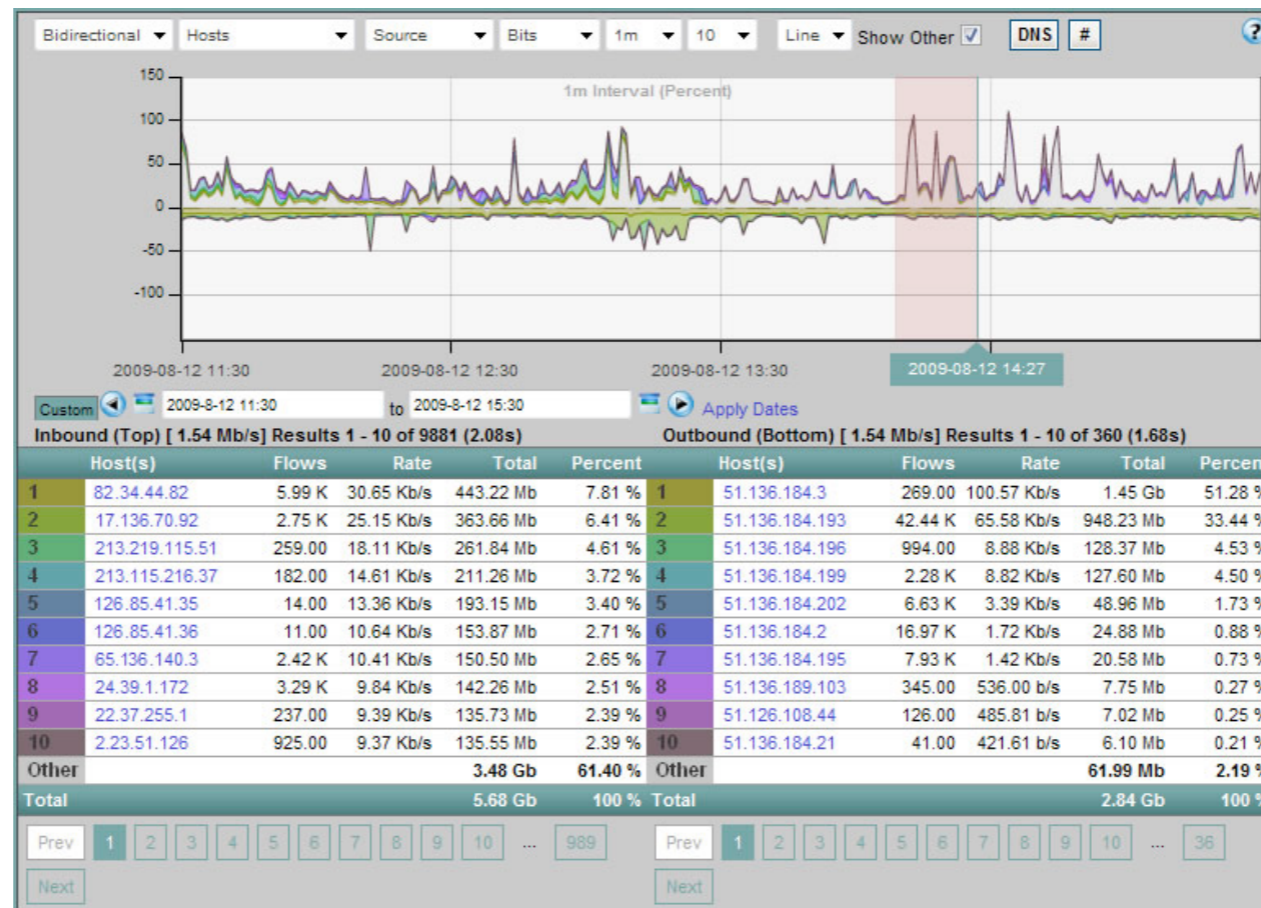
concerned that changes would
cause problems with existing
functionality

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

Ensuring correct operations today

Manual spot-checking (pings, traceroutes)

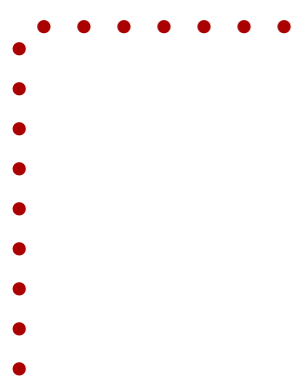
Monitoring of events & flows



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

Network Verification

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



Configuration

Configuration verification

Control software

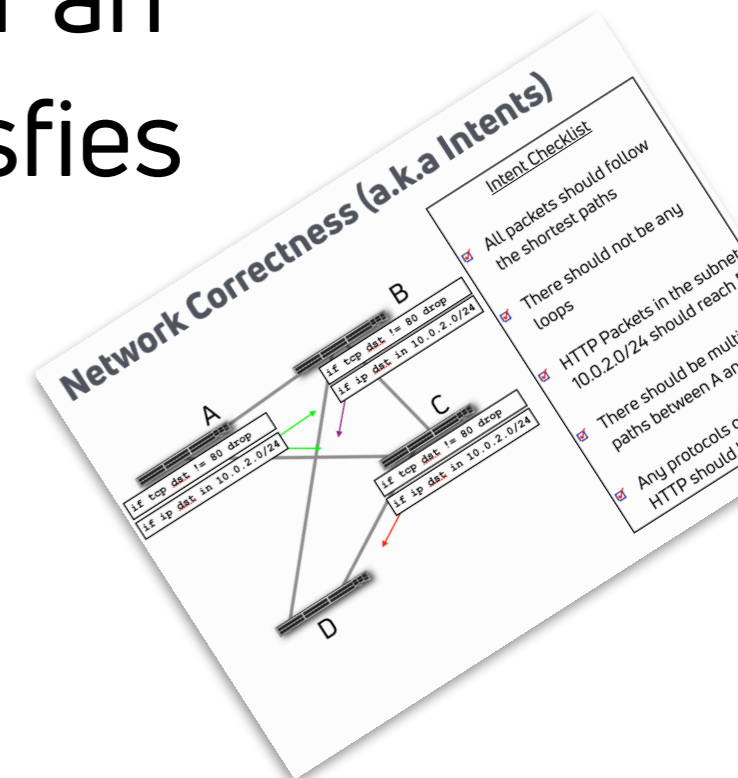
Controller verification & verifiable control languages

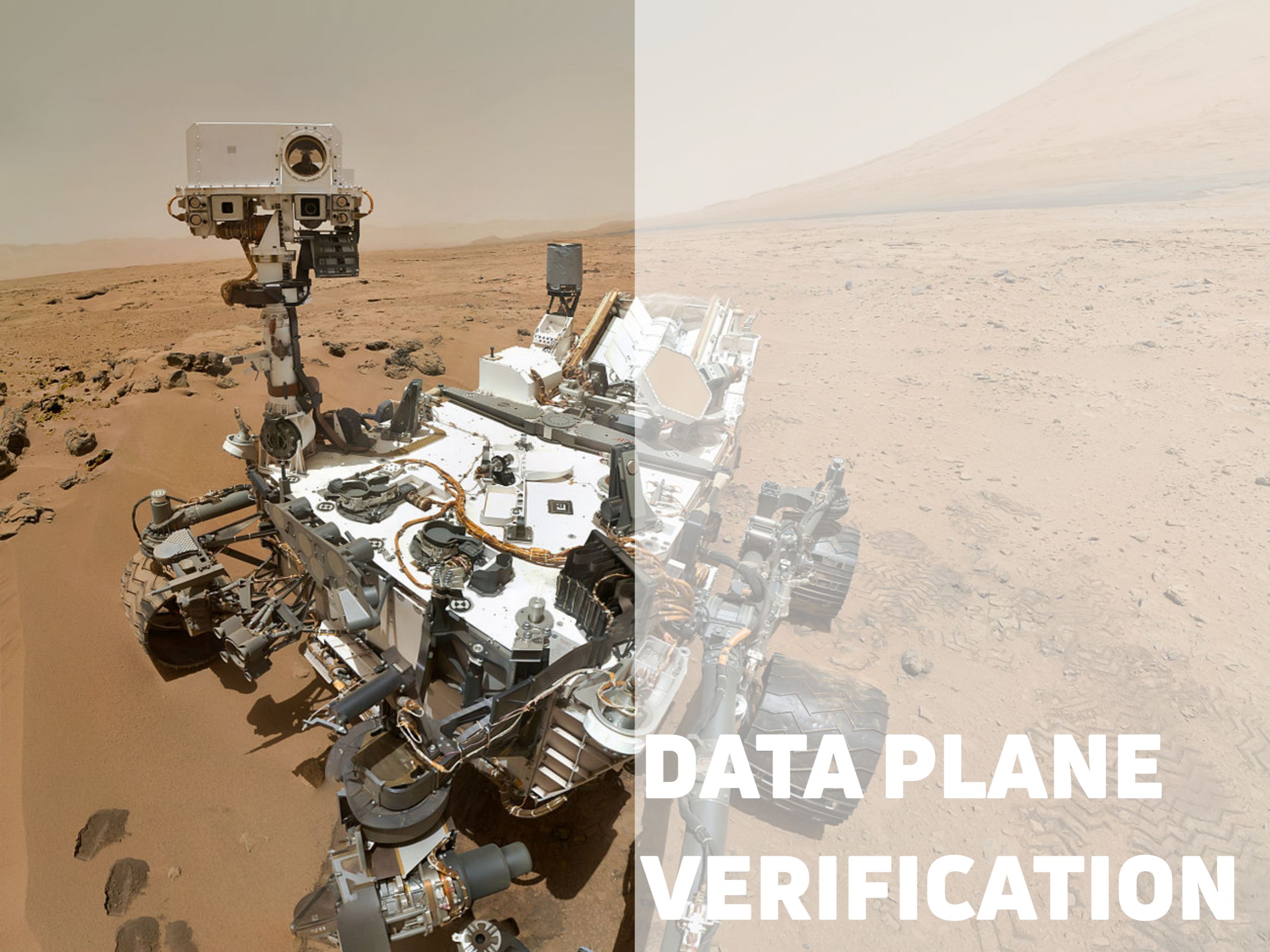
Data plane state

Data plane verification

Packet processing

Software switch verification, Formal P4 semantics





**DATA PLANE
VERIFICATION**

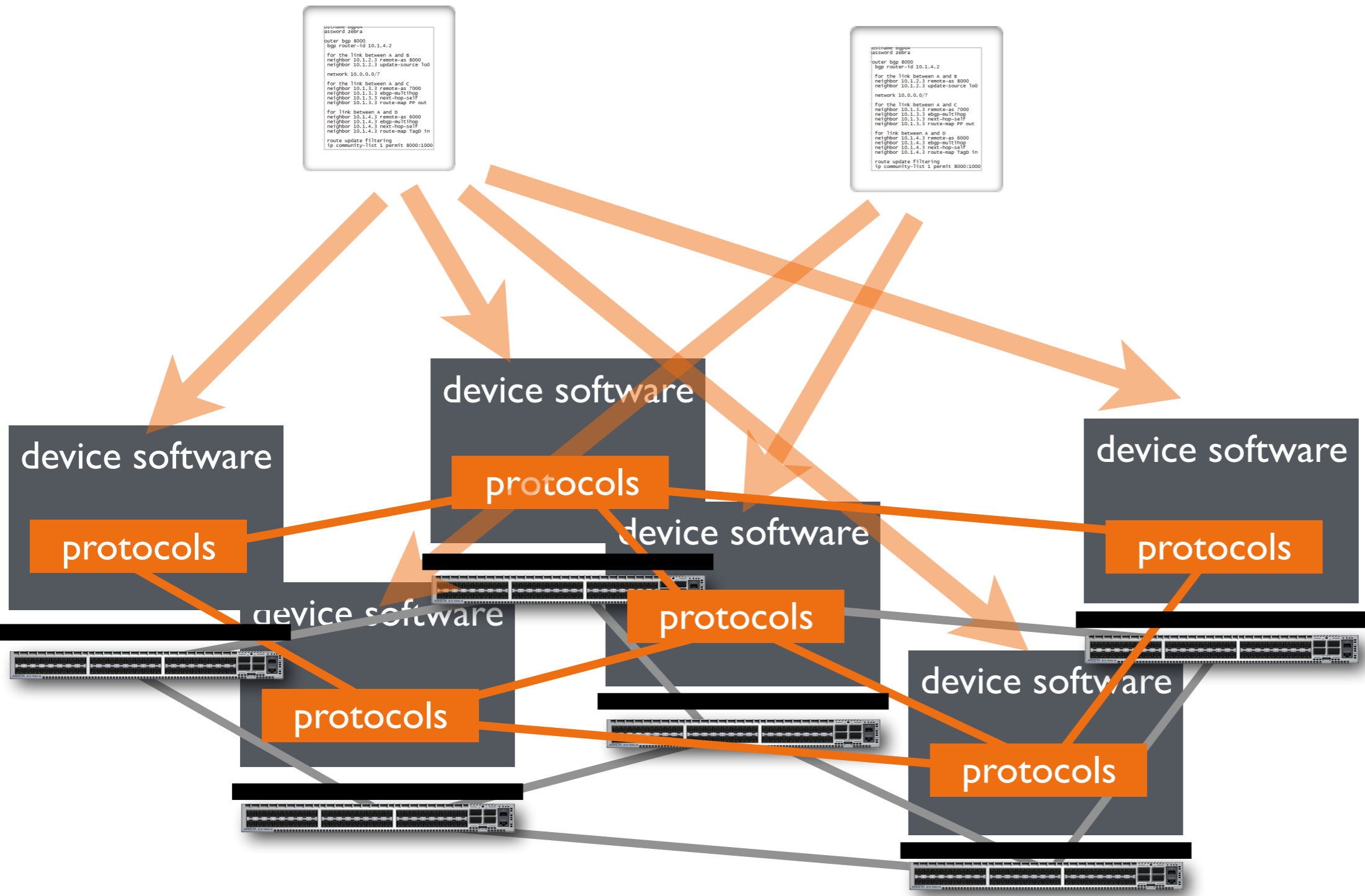
Configuration verification

```
hostname bgpasa
password ZEBRA
router bgp 8000
  bgp router-id 10.1.1.4.2
  for the link between A and B
  neighbor 10.1.2.3 remote-as 8000
  neighbor 10.1.2.3 update-source loop
  network 10.0.0.0/7
  for the link between A and C
  neighbor 10.1.3.3 remote-as 7000
  neighbor 10.1.3.3 ebgp-multihop
  neighbor 10.1.3.3 next-hop-self
  neighbor 10.1.3.3 route-map PP out
  for link between A and D
  neighbor 10.1.4.3 remote-as 6000
  neighbor 10.1.4.3 ebgp-multihop
  neighbor 10.1.4.3 next-hop-self
  neighbor 10.1.4.3 route-map Tagp in
  route update filtering
  ip community-list 1 permit 8000:1000
```

```
hostname bgpbc
password ZEBRA
router bgp 8000
  bgp router-id 10.1.1.4.2
  for the link between A and B
  neighbor 10.1.2.3 remote-as 8000
  neighbor 10.1.2.3 update-source loop
  network 10.0.0.0/7
  for the link between A and C
  neighbor 10.1.3.3 remote-as 7000
  neighbor 10.1.3.3 ebgp-multihop
  neighbor 10.1.3.3 next-hop-self
  neighbor 10.1.3.3 route-map PP out
  for link between A and D
  neighbor 10.1.4.3 remote-as 6000
  neighbor 10.1.4.3 ebgp-multihop
  neighbor 10.1.4.3 next-hop-self
  neighbor 10.1.4.3 route-map Tagp in
  route update filtering
  ip community-list 1 permit 8000:1000
```

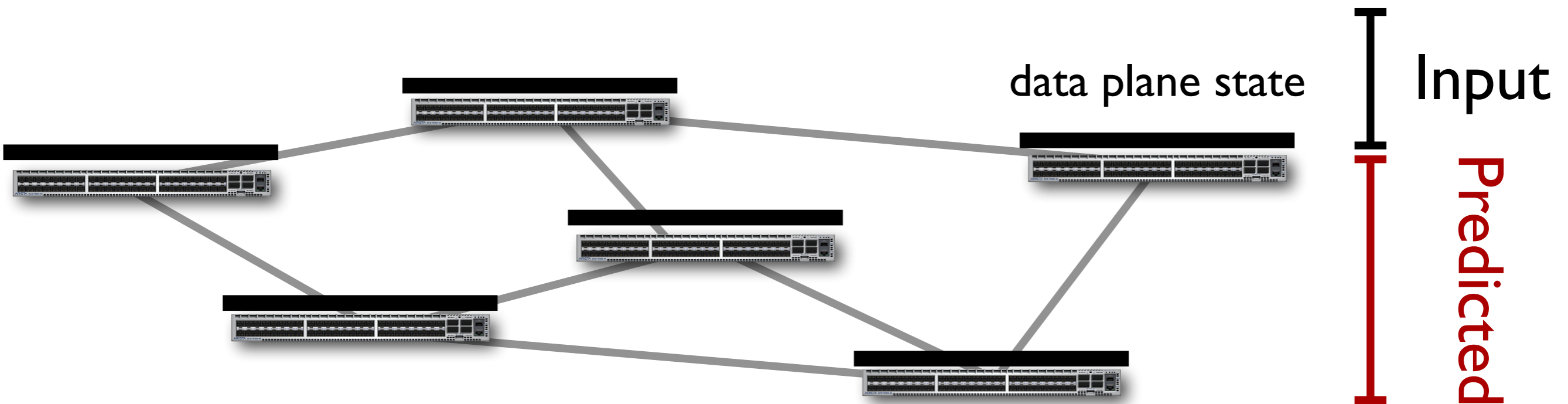
Input

Predicted



Data plane verification

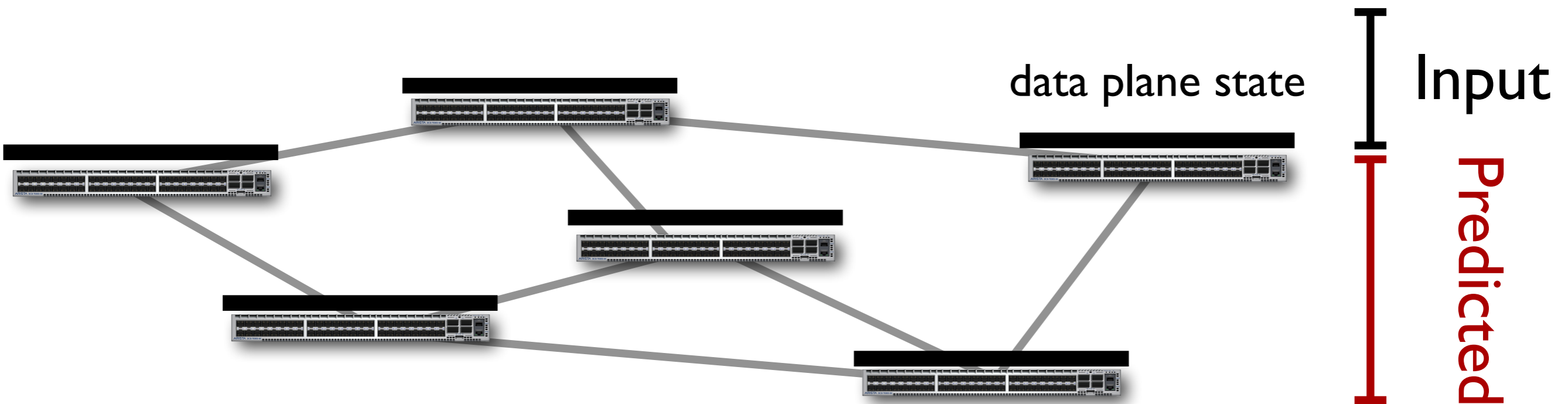
Verify the network
as close as possible
to its actual behavior



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"

Need for accuracy

2-year study of O(100) incidents in Microsoft Azure cloud

(Configuration Verification)

Root Cause	Proportion	Examples	CrystalNet Coverage	Verification Coverage
Software Bugs	36%	bugs in routers, middleboxes, management tools	✓	X
Config. Bugs	27%	wrong ACL policies, traffic black holes, route leaking	✓	✓
Human Errors	6%	mis-typing, unexpected design flaws	✓	X
Hardware Failures	29%	ASIC driver failures, silent packet drops, fiber cuts, power failures	X	X
Unidentified	2%	transient failures	X	X

Table 1: Root causes of O(100) significant and customer-impacting incidents in our network (2015 - 2017).

Example software bug

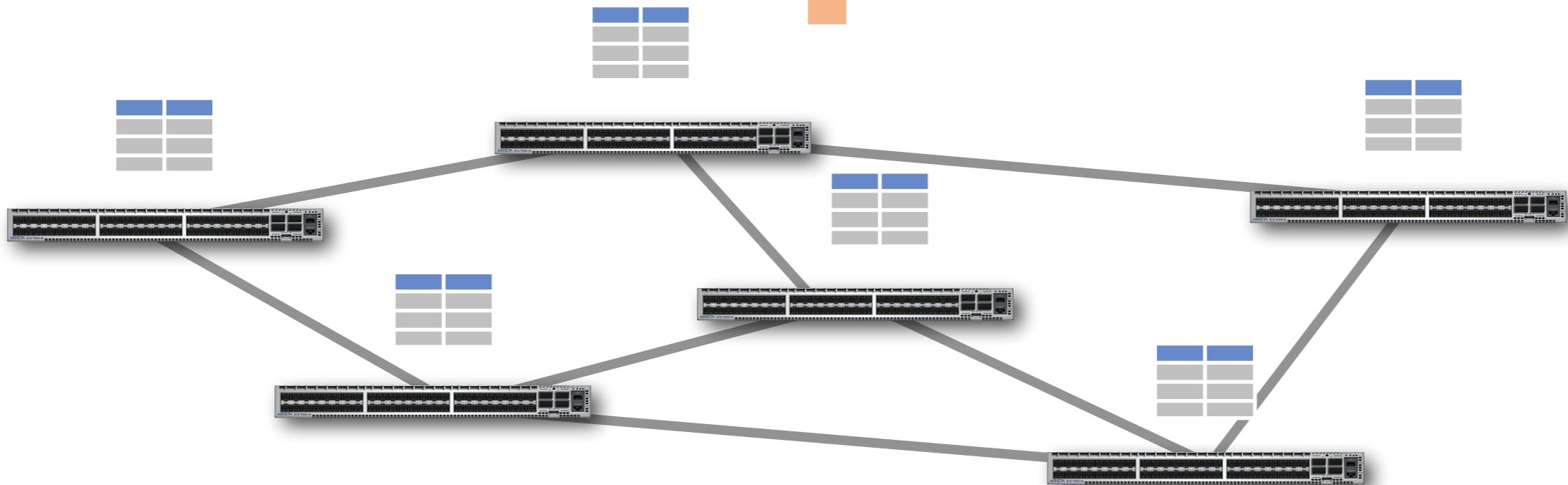
- “a vendor changed the format of ACLs in the new release, but neglected to document the change clearly. As a result, the old configuration files were processed incorrectly by switches running the new firmware.”

Architecture

“Can any packet starting at A reach B?”


Diagnosis

Verifier



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)?

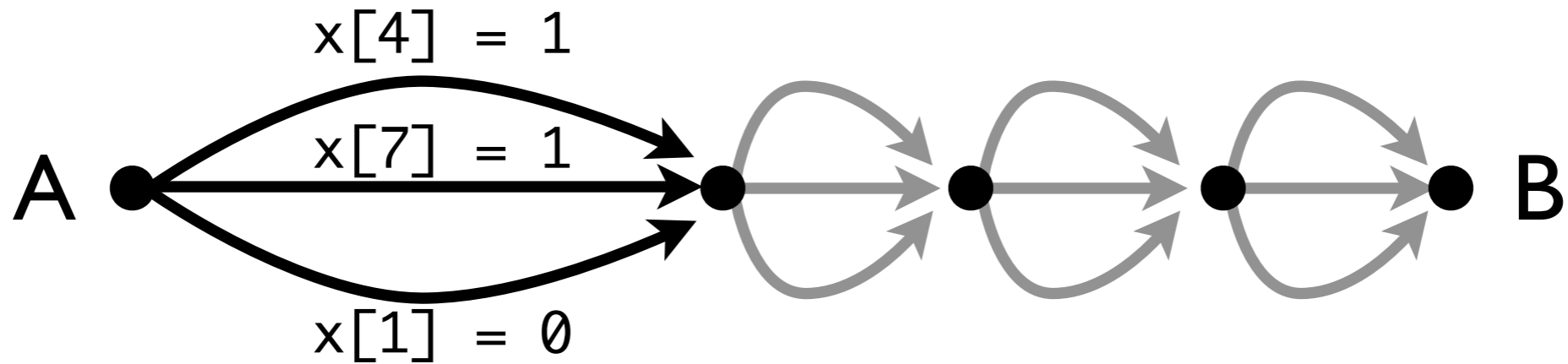
- Undecidable in general
- EXPSPACE-complete with reasonable assumptions
- Easier with additional assumptions

**Some complexity results for
stateful network verification**

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

A-to-B query with bitmask

Packet: $x[0] x[1] x[2] \dots x[n]$



$$(x_4 \vee x_7 \vee \bar{x}_1) \wedge (\dots) \wedge (\dots) \wedge (\dots)$$

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



$$P(u, v) = \text{dst_ip} \in 10.1.1.0/24$$

Reachability as SAT solving

Goal: reachability from u to w

==

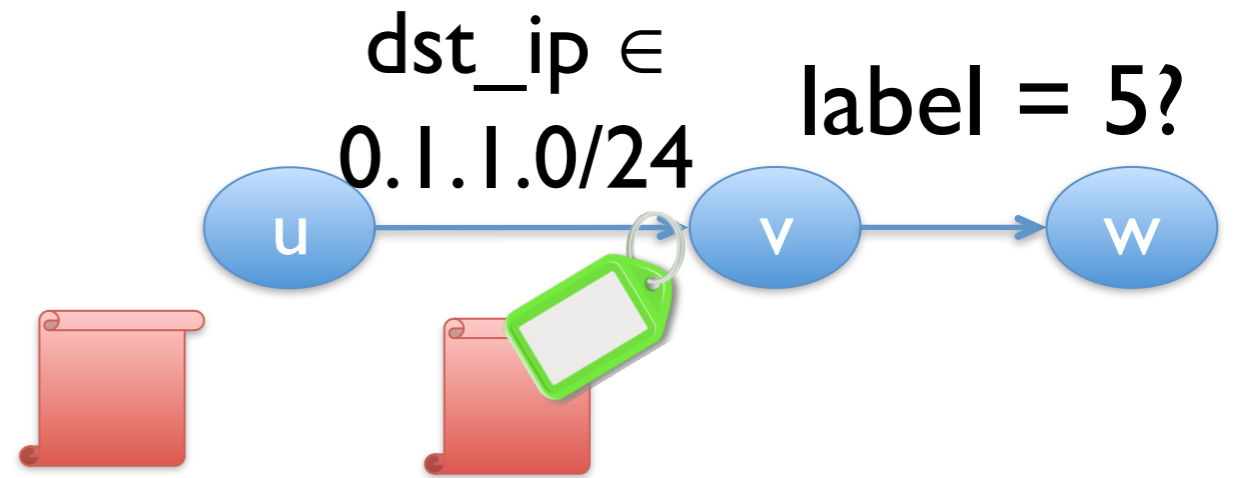


$C = (P(u, v) \wedge P(v, w))$ is satisfiable

- 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 protocols like MPLS, QoS, NAT, etc.



- Model the history of packets: vector over time
- Packet transformation \Rightarrow 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

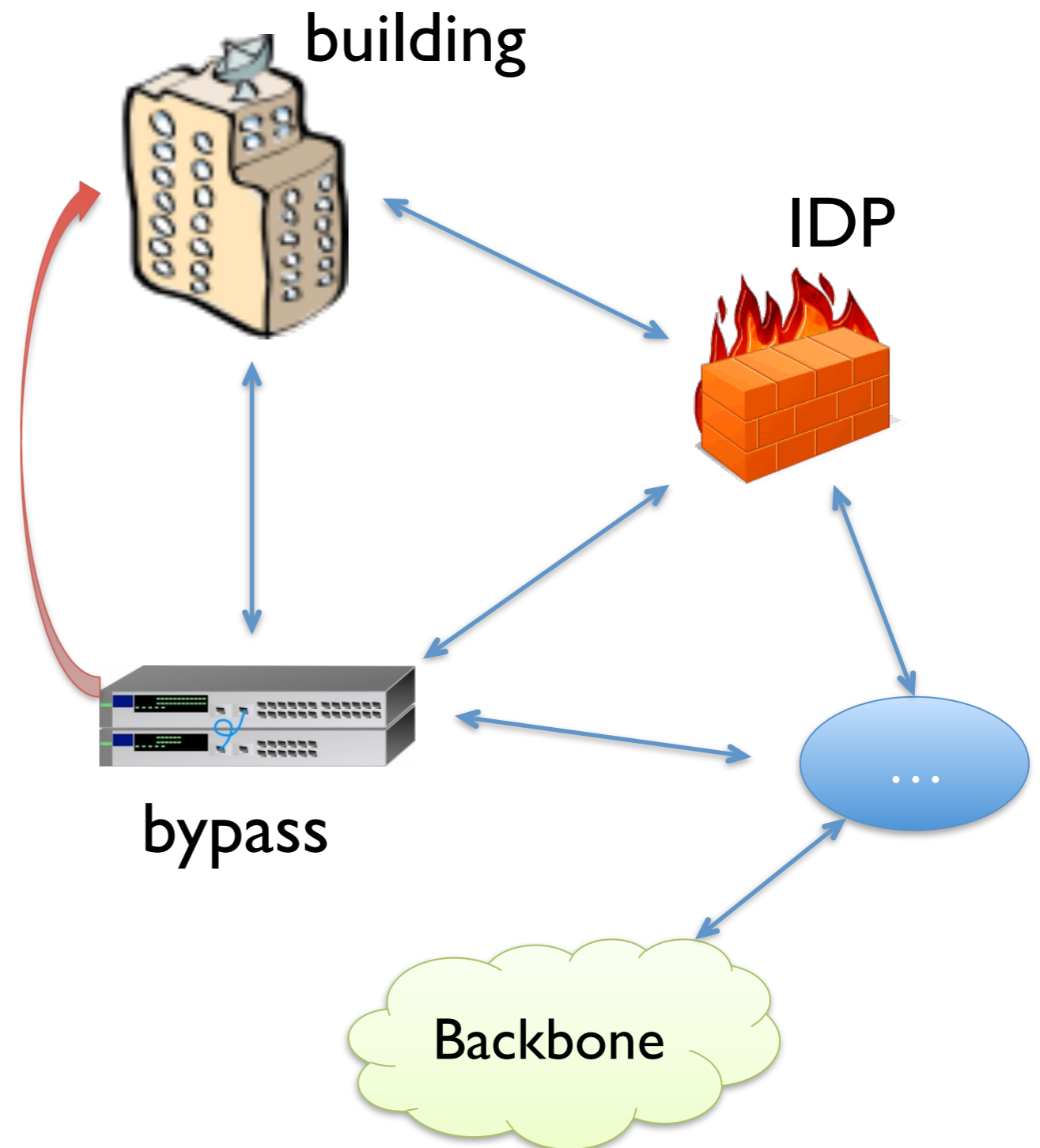
	Loop	Packet loss	Consistency
Being fixed	9	0	0
Stale config.	0	13	1
Total alerts	9	17	2

Forwarding loops

IDP was overloaded,
operator introduced
bypass

Bypass routed campus
traffic to IDP through static
routes

Introduced 9 loops



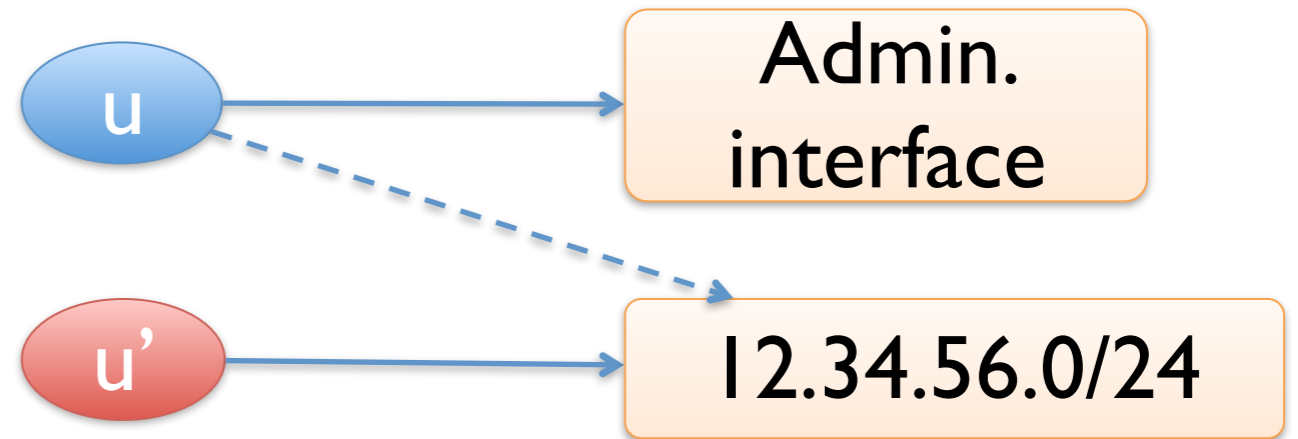
Multiple policy violations found

Packet loss

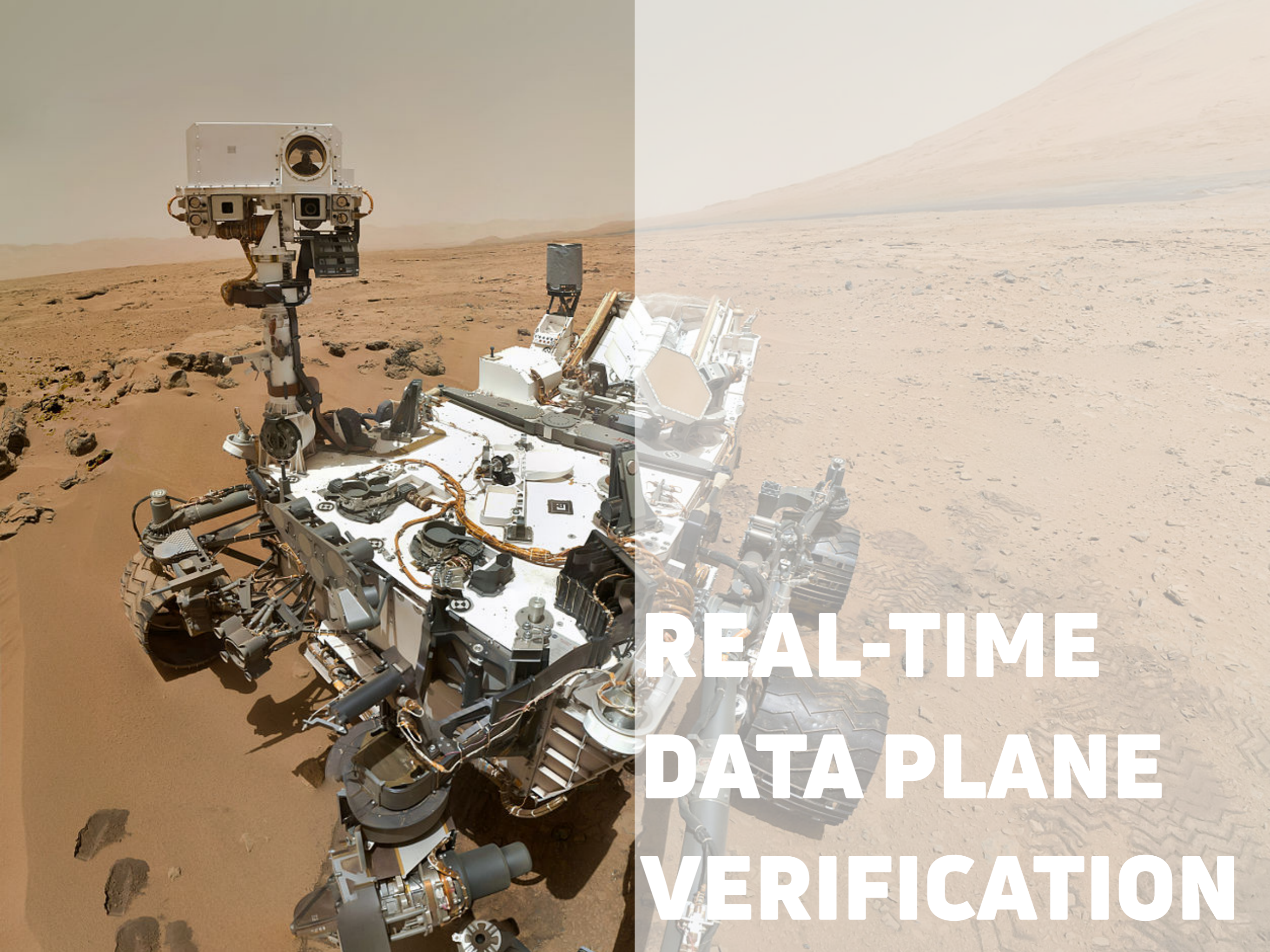


- Blocking compromised machines at IP level
- Stale configuration 2+ years old

Consistency



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



**REAL-TIME
DATA PLANE
VERIFICATION**

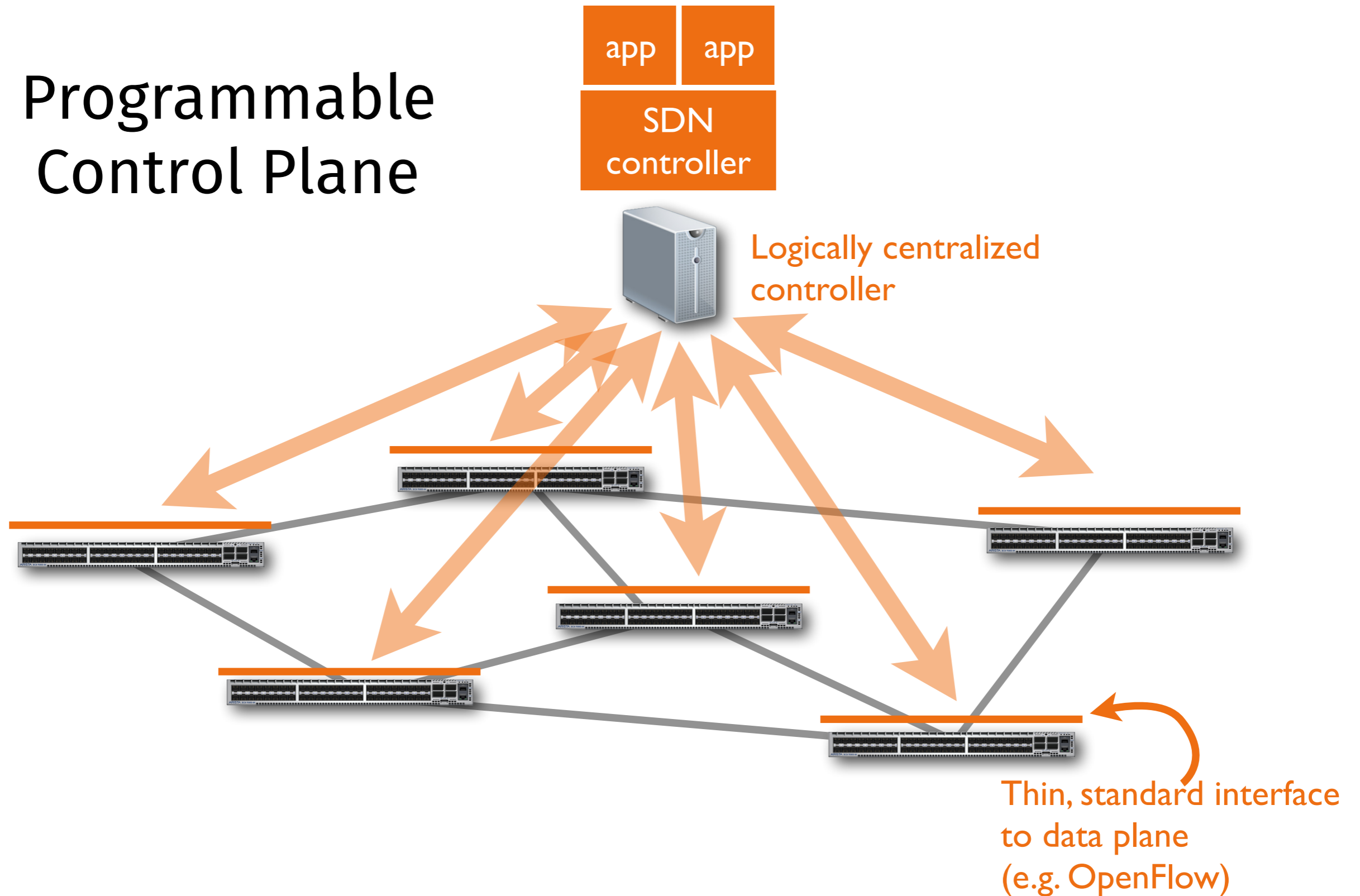
Not so simple

Challenge #1: Obtaining real time view

Challenge #2: Verify quickly

SDN: Control plane programmability

Programmable
Control Plane

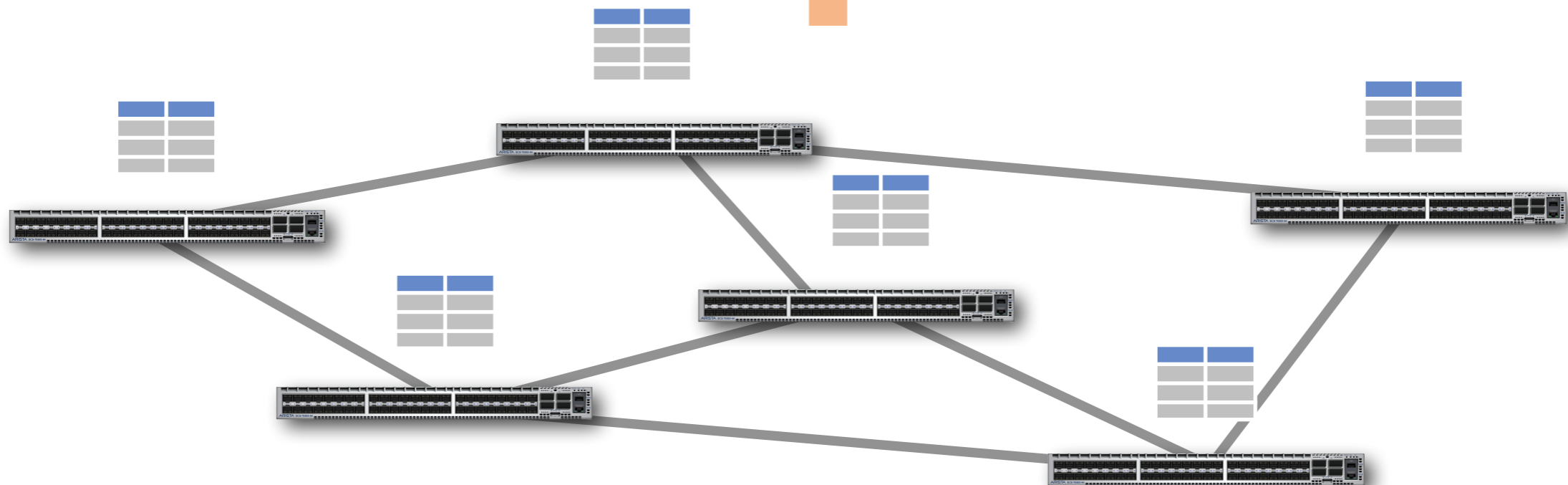


Data Plane Verification Architecture

“Service S reachable only through firewall?”

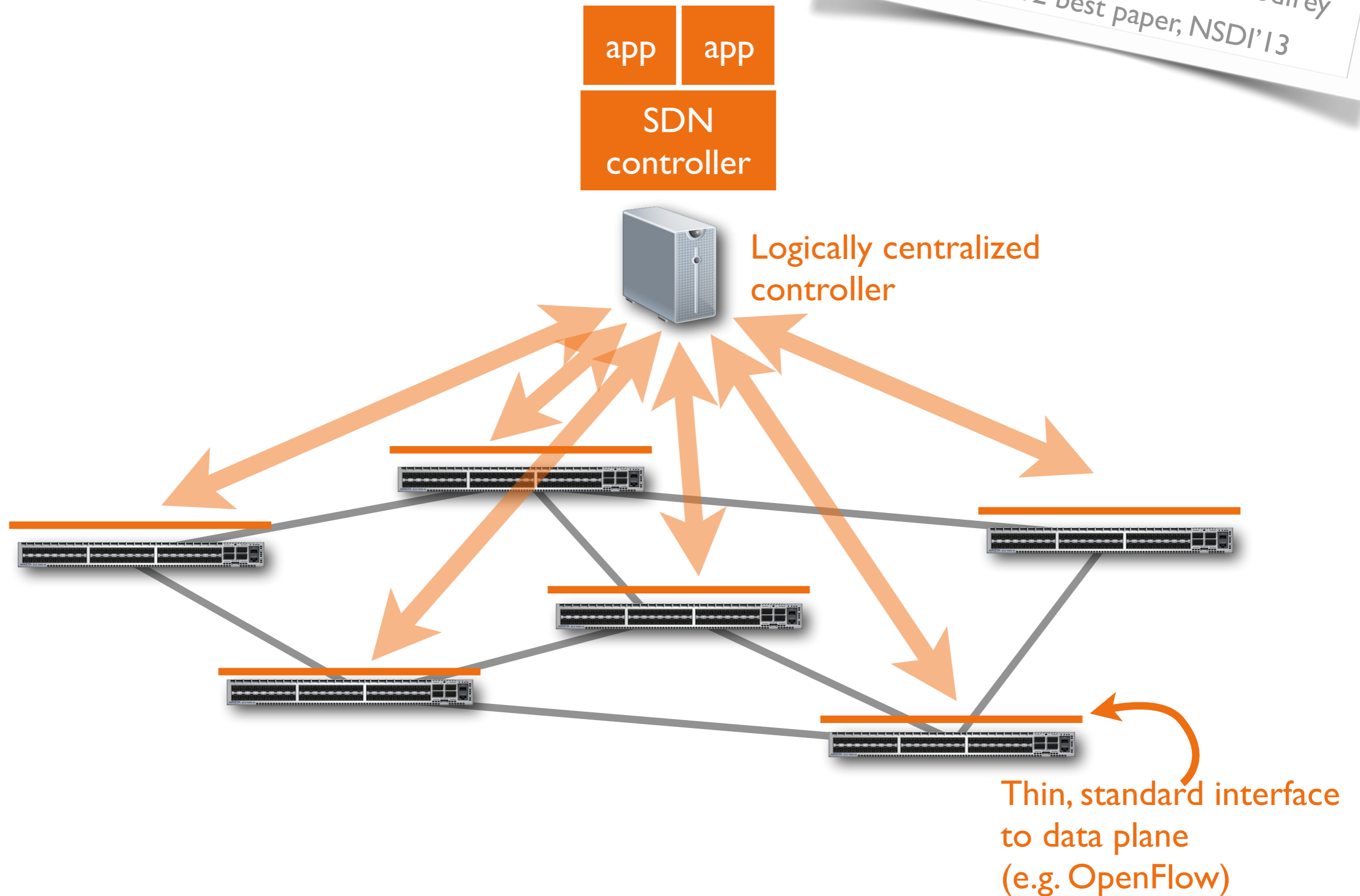
Diagnosis

Verifier



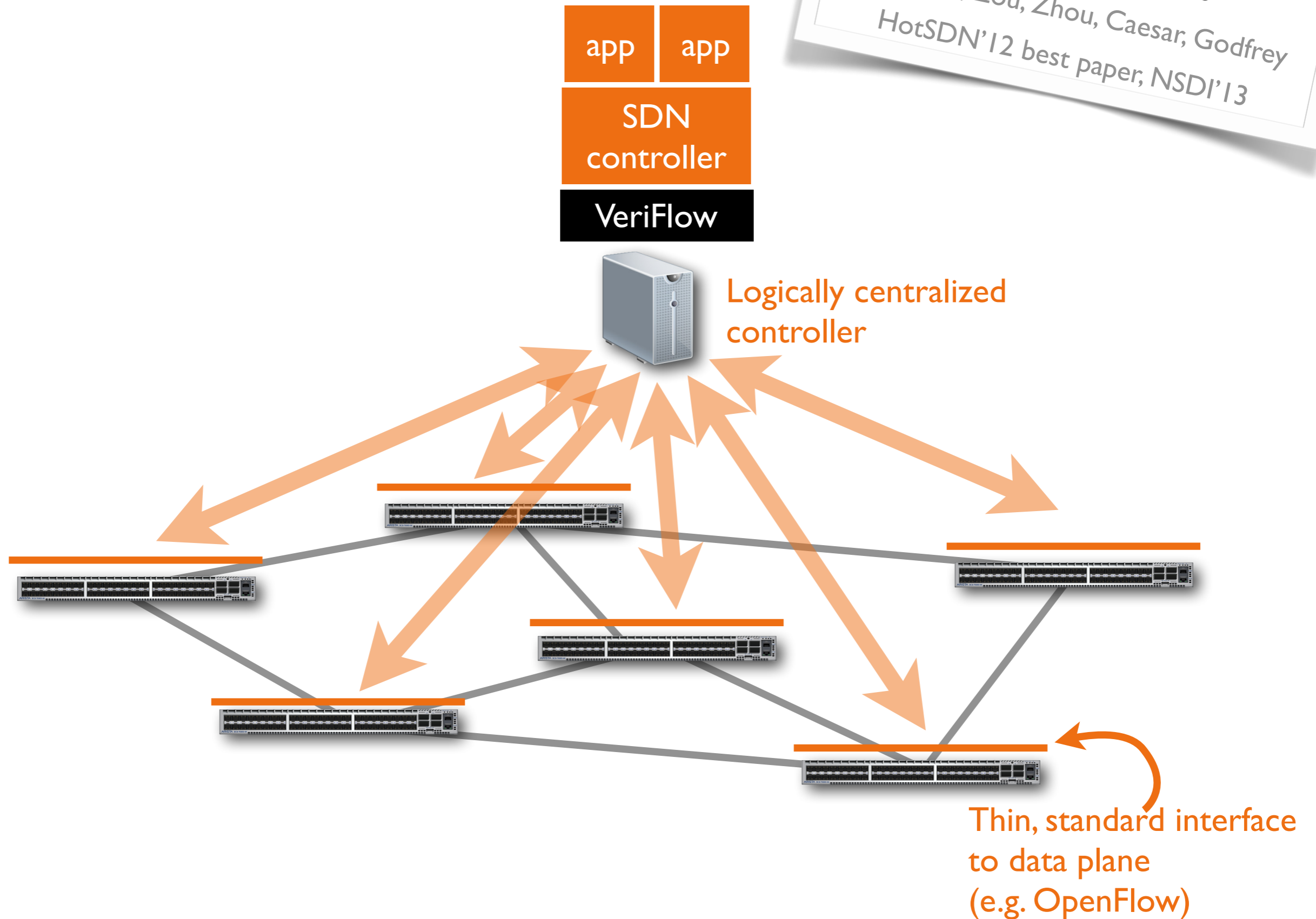
VeriFlow architecture

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

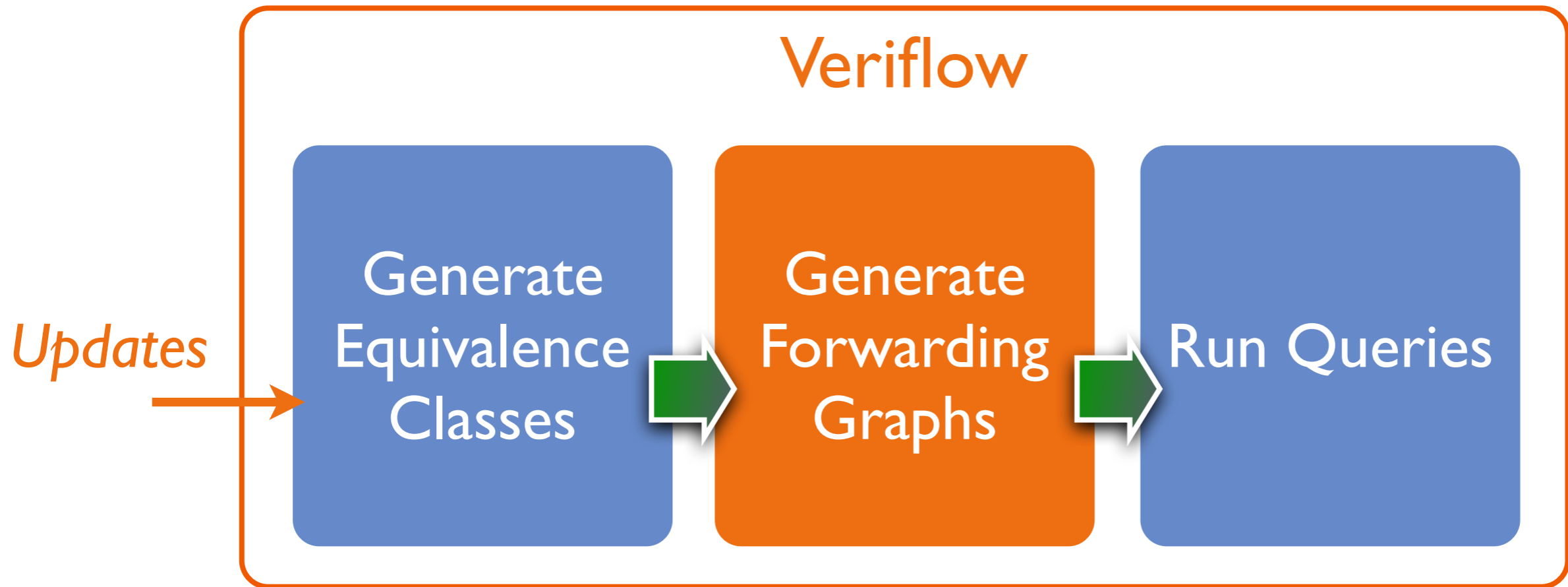


VeriFlow architecture

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



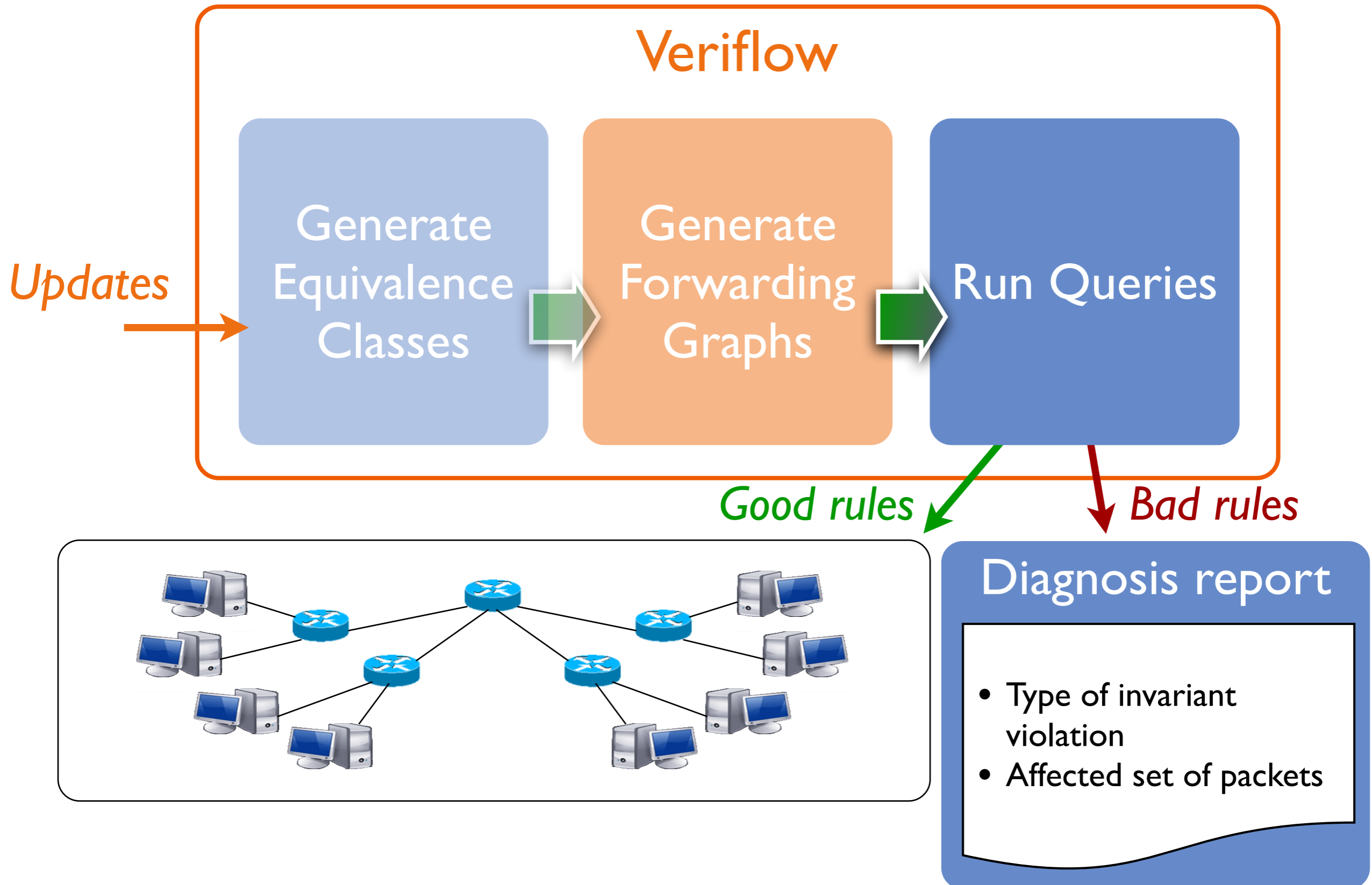
Verifying invariants quickly



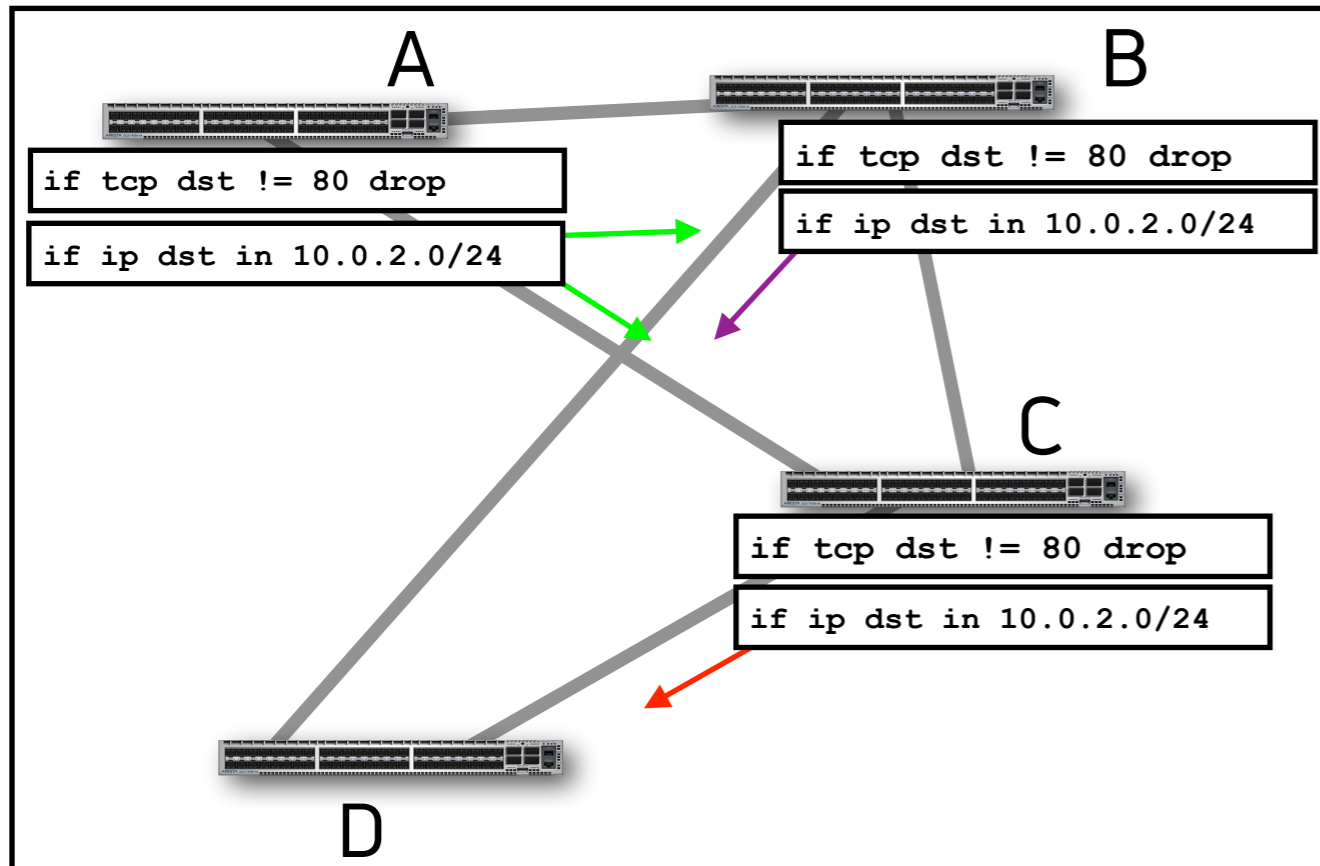
Overview:

1. Limit the search space
2. Represent forwarding behaviors using graphs
3. Run Light-weight graph-based algorithm to check

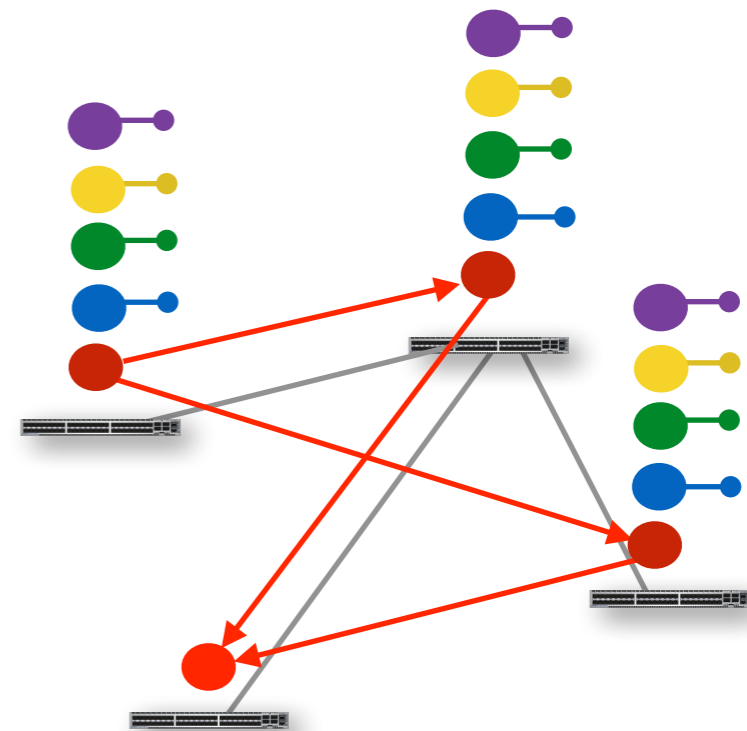
Verifying invariants quickly



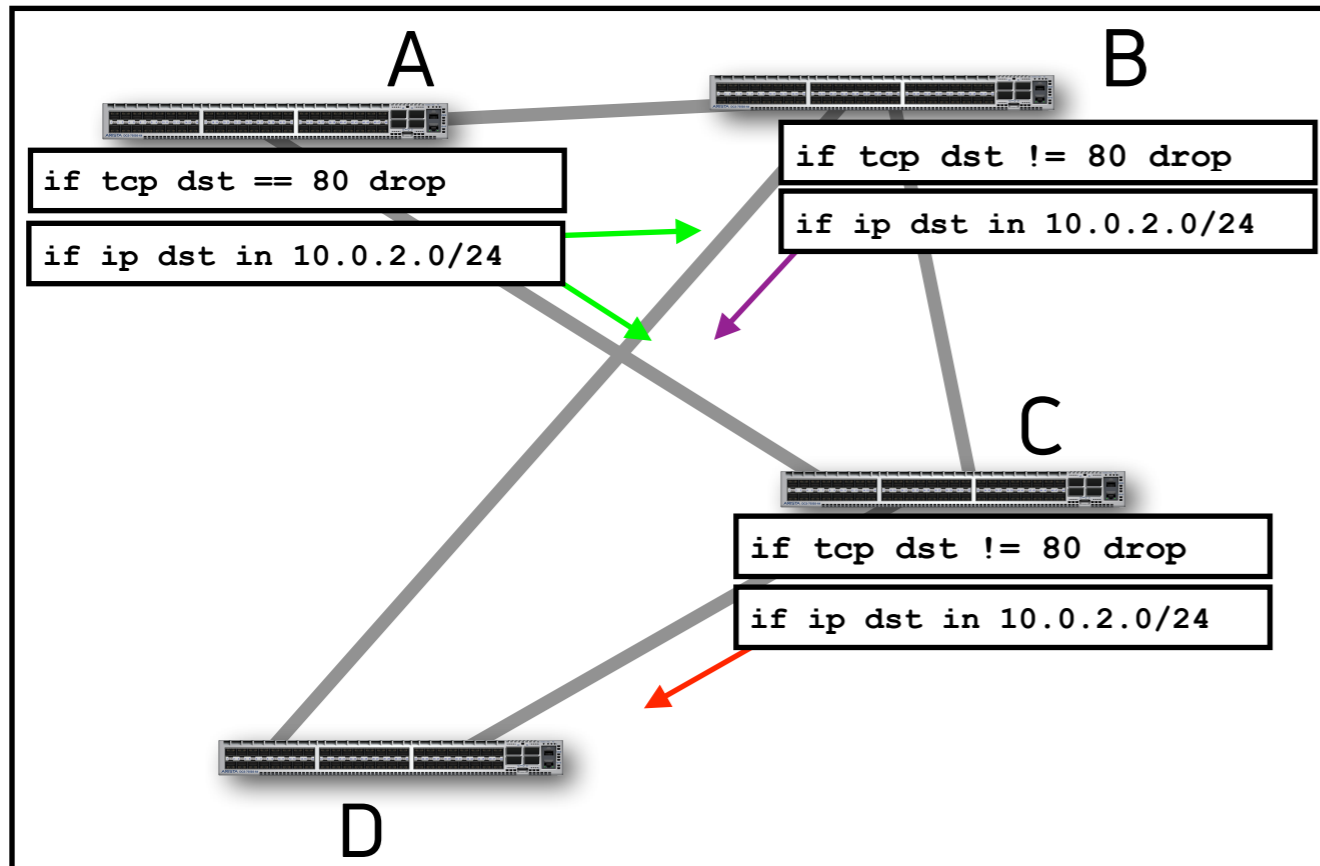
Forwarding Graphs



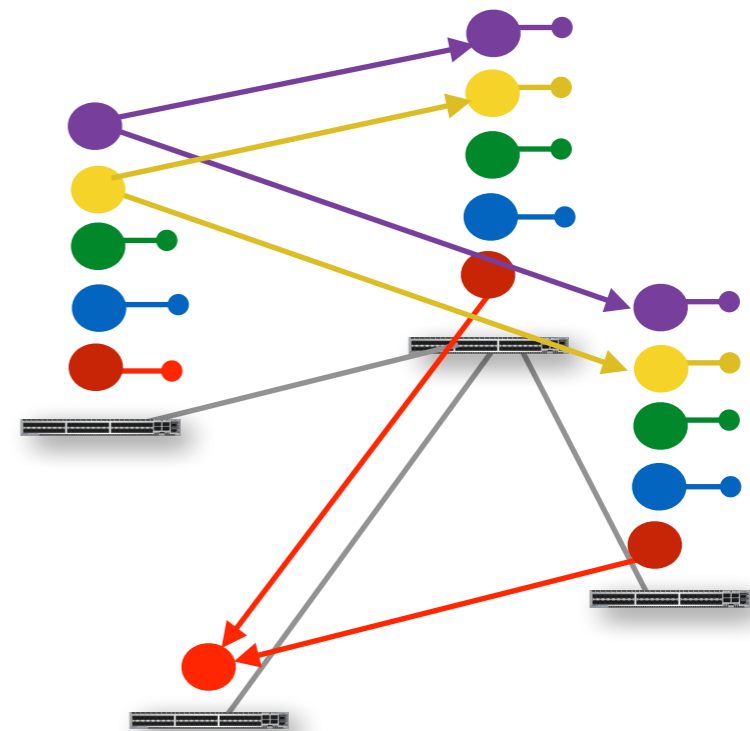
ip dst [0, 167772671], tcp dst [0, 65535]
ip dst [167772672, 167772927], tcp dst [80, 80]
ip dst [167772928, 4294967295], tcp dst [0, 65535]
ip dst [167772672, 167772927], tcp dst [0, 79]
ip dst [167772672, 167772927], tcp dst [81, 65535]



Forwarding Graphs



ip dst [0, 167772671], tcp dst [0, 65535]
ip dst [167772672, 167772927], tcp dst [80, 80]
ip dst [167772928, 4294967295], tcp dst [0, 65535]
ip dst [167772672, 167772927], tcp dst [0, 79]
ip dst [167772672, 167772927], tcp dst [81, 65535]



Intent API

What intents can you check?

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

Veriflow's API enables custom query algorithms

- Full access to forwarding graph
- For incremental verification: Gives access to the "diff": forwarding subgraph affected by an update from the SDN controller
- Verification becomes a standard graph traversal algorithm

Evaluation

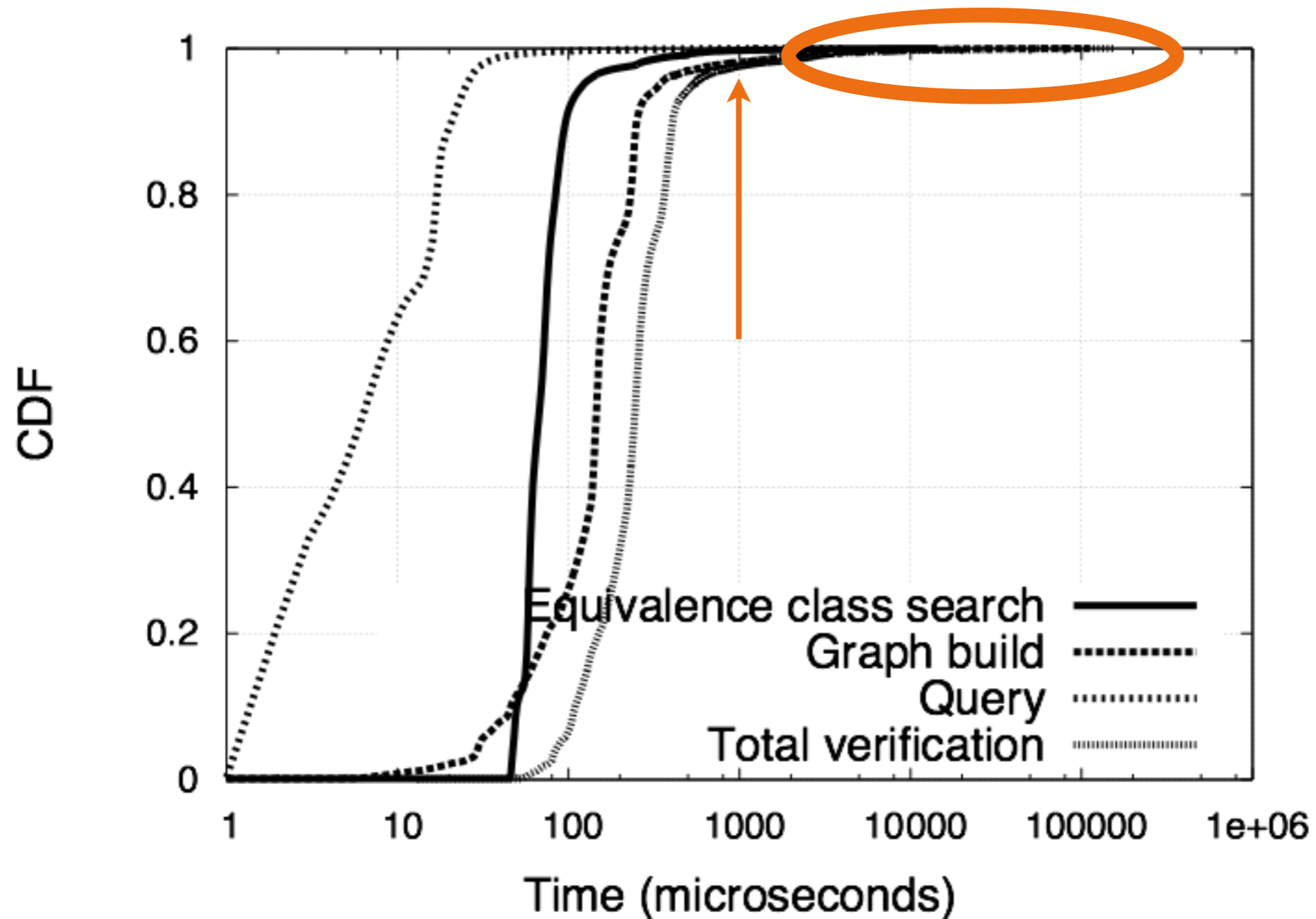
Simulated network

- Real-world BGP routing tables (RIBs) from *RouteViews* totaling 5 million RIB entries
- Injected into 172-router network (Ebone ISP)

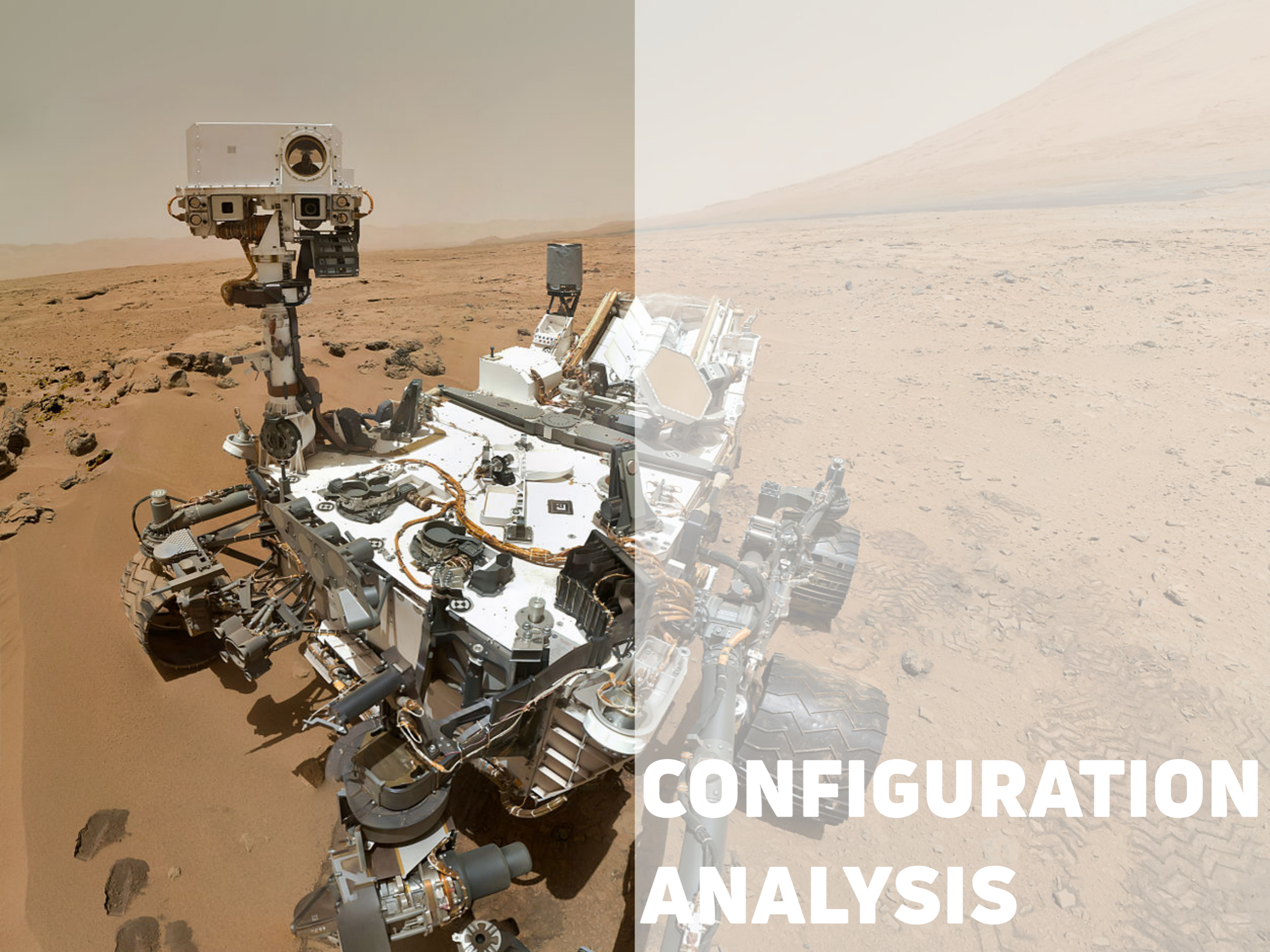
Measure time to process each forwarding change

- 90,000 updates from *RouteViews*
- Check for loops and black holes

Microbenchmark latency

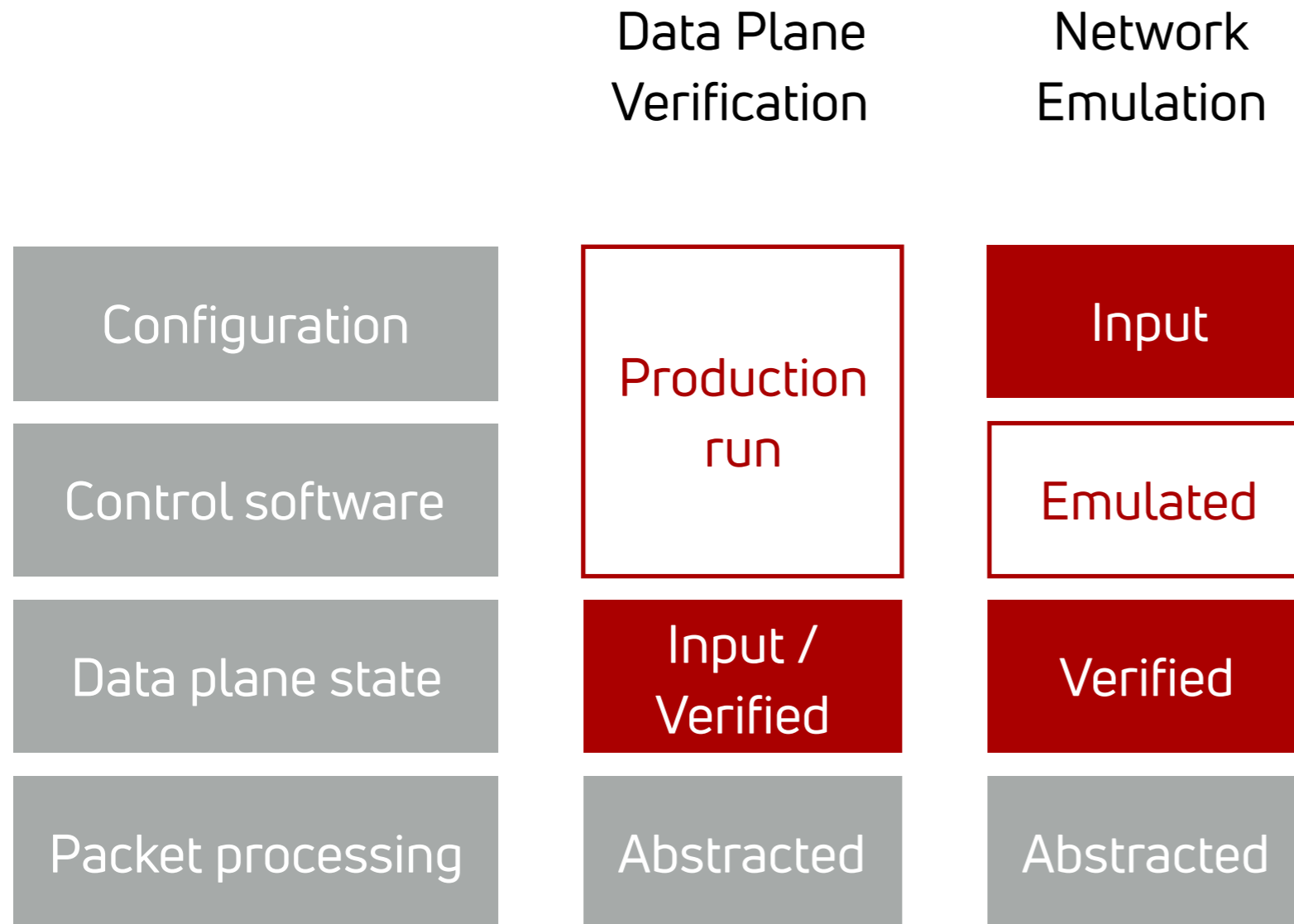


97.8% of updates verified within 1 ms

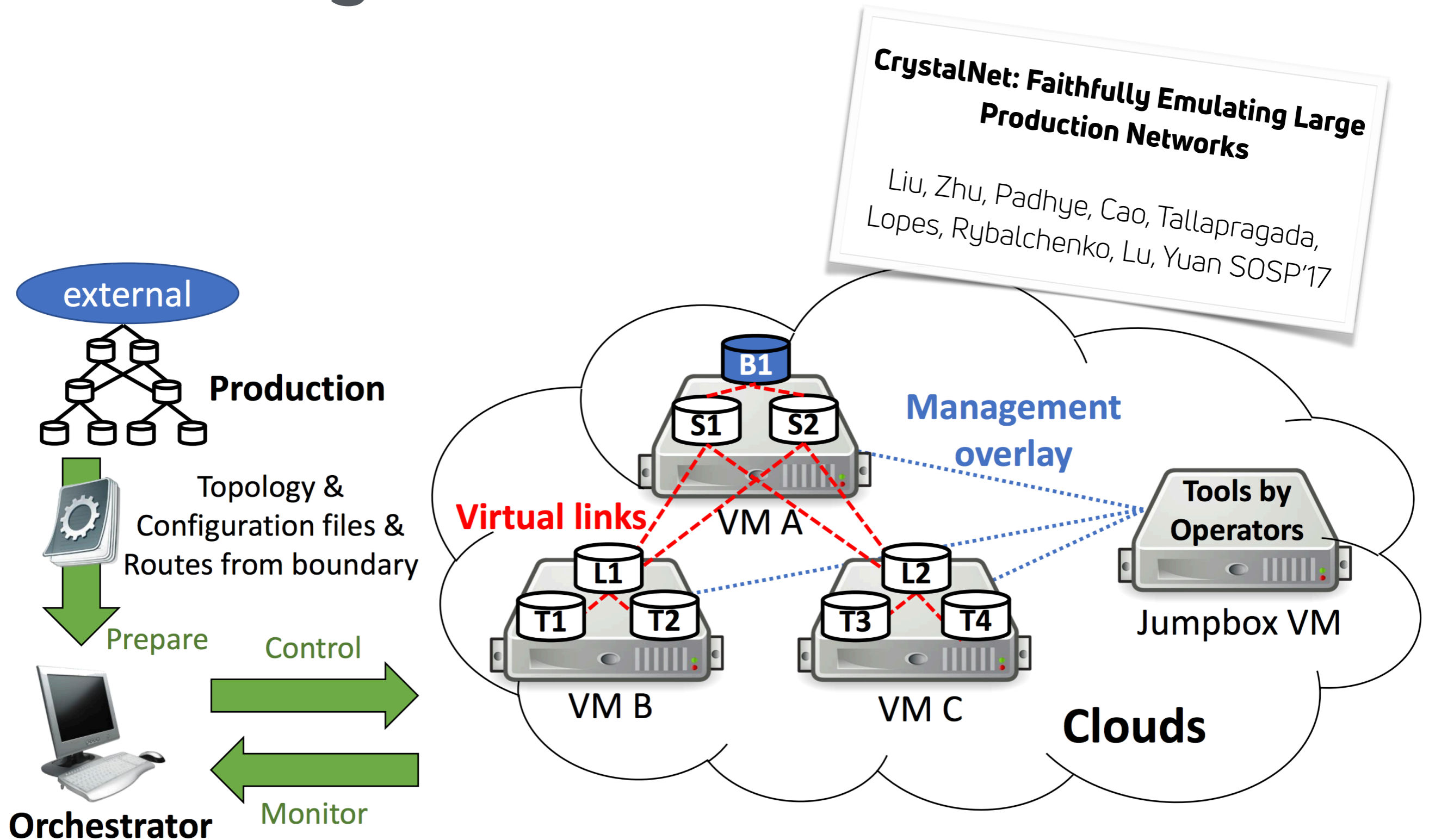


CONFIGURATION ANALYSIS

Landscape of approaches



Emulating Production Networks




Landscape of approaches

	Data Plane Verification	Network Emulation	Control plane simulation
Configuration		Input	Input
Control software	Production run	Emulated	Simulate a model run
Data plane state	Input / Verified	Verified	Verified
Packet processing	Abstracted	Abstracted	Abstracted

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

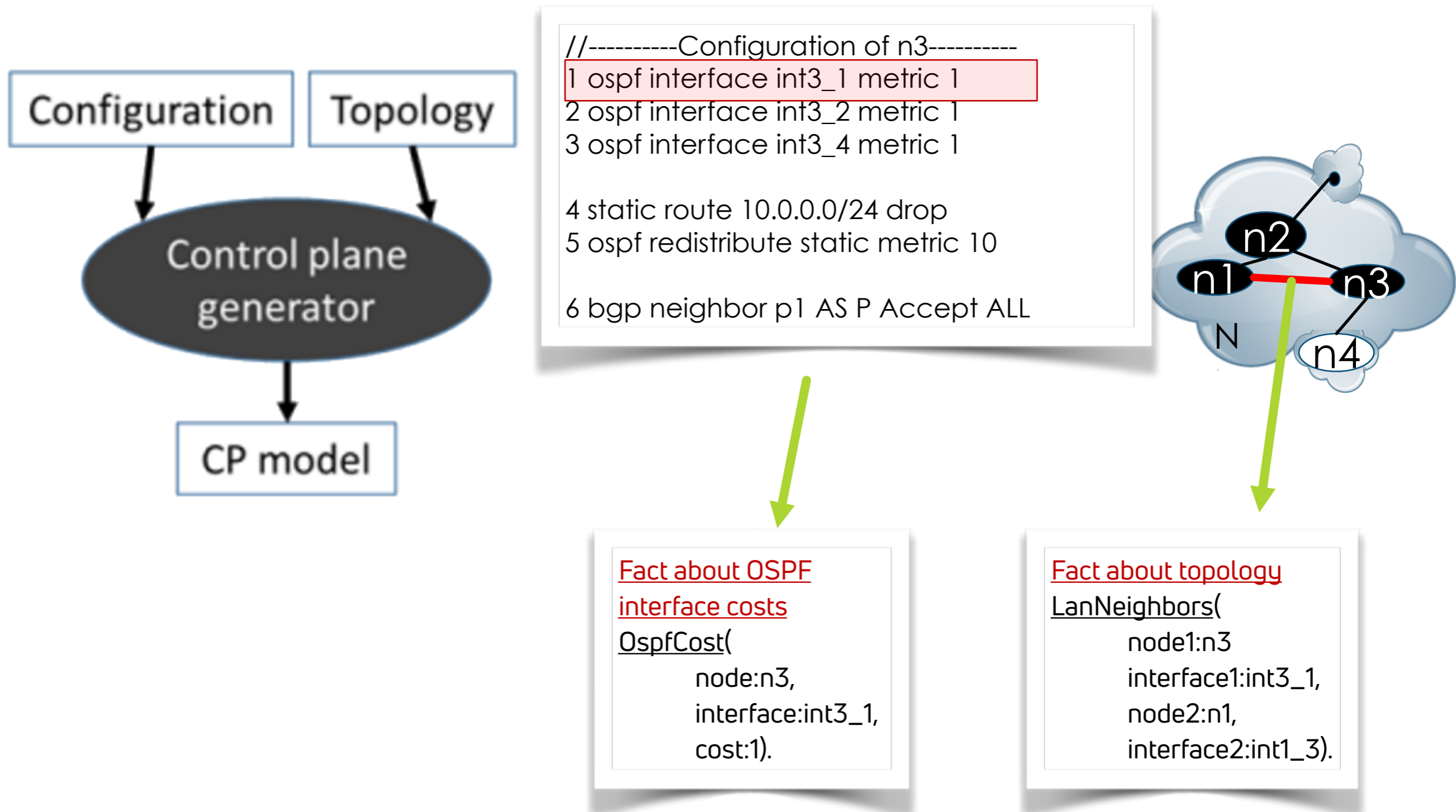
Available at <http://www.batfish.org>

Has found real bugs in real networks

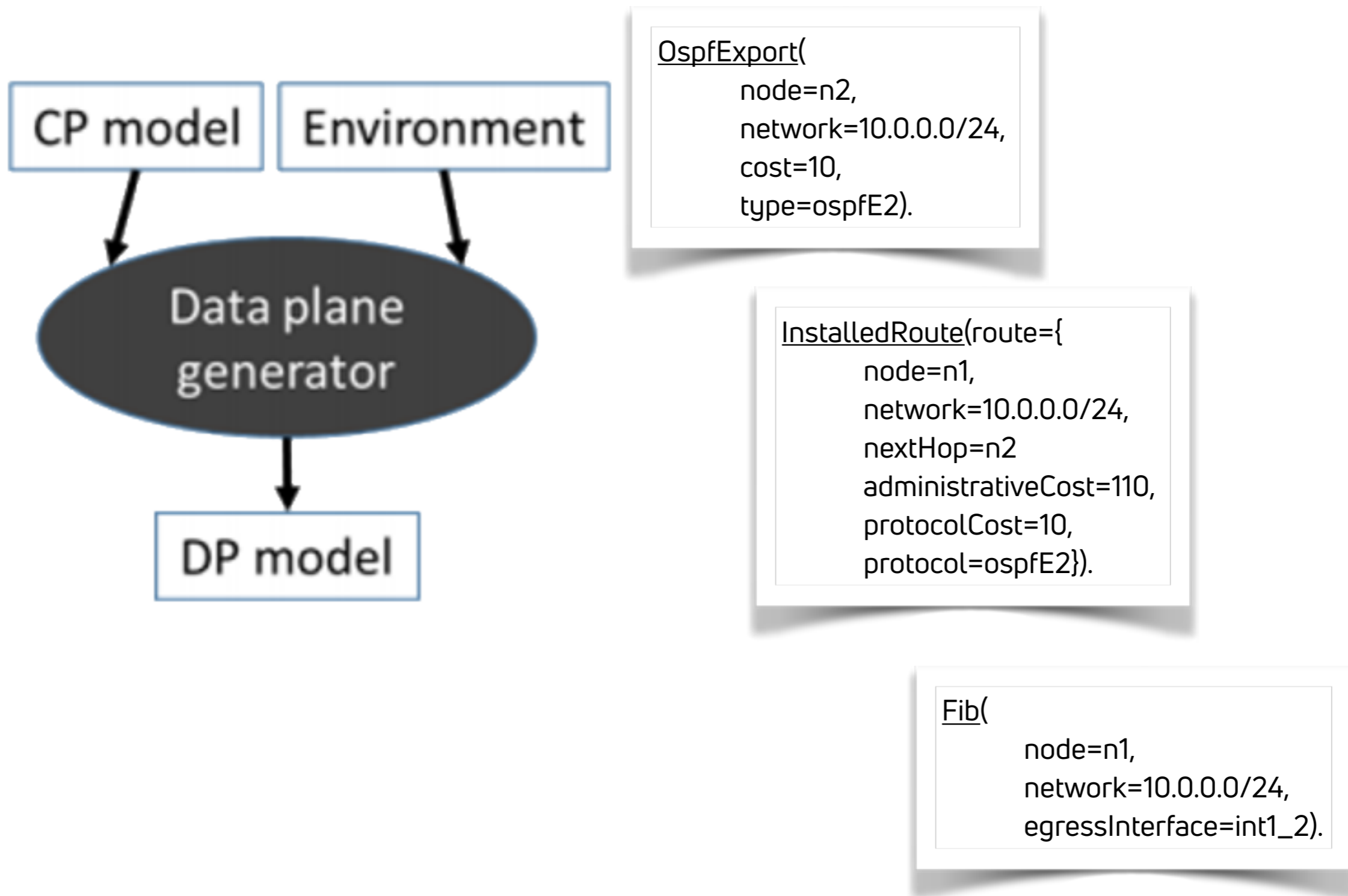
4 stages:

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

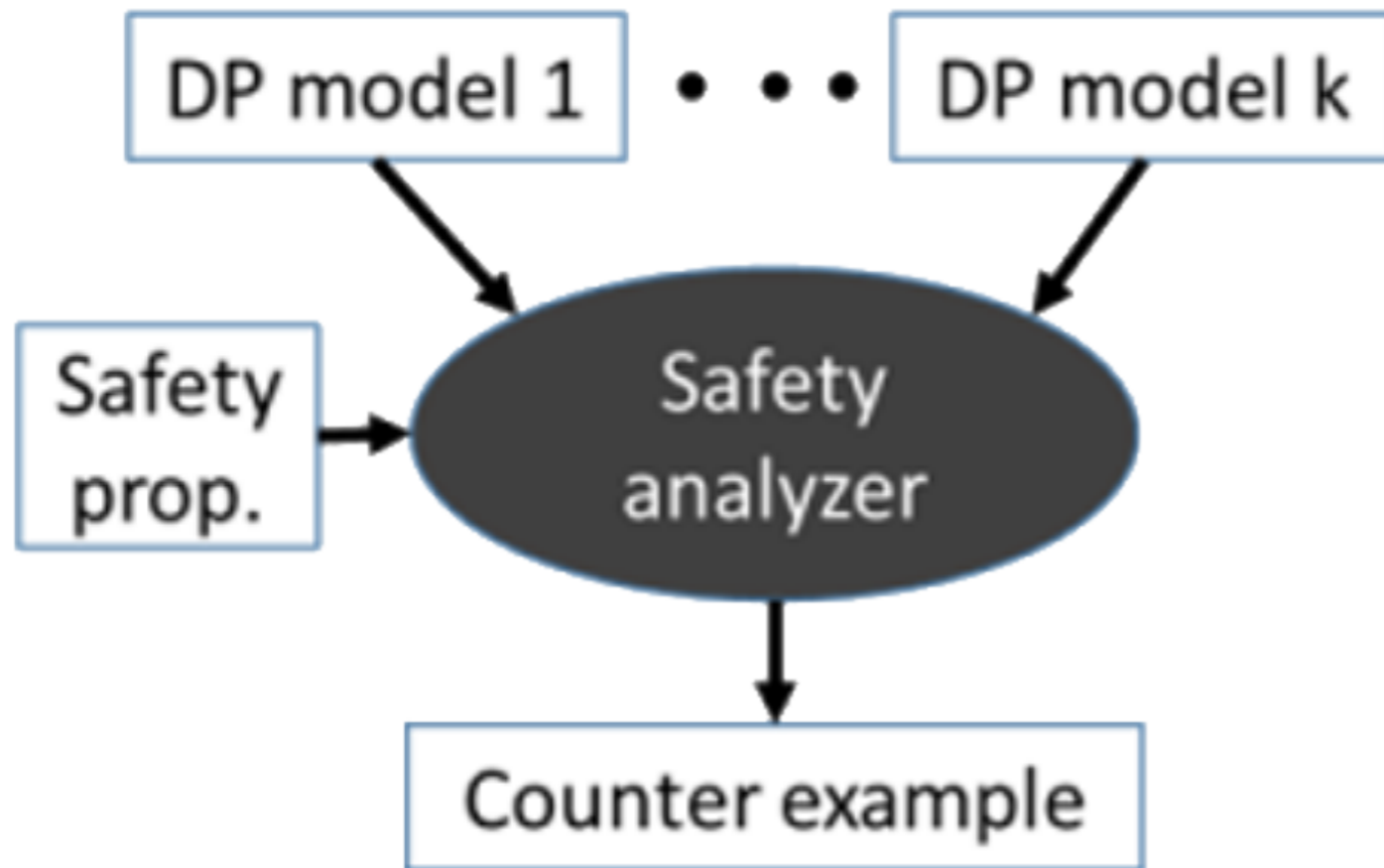
Stage 1: Extract control plane model



Stage 2: Compute data plane



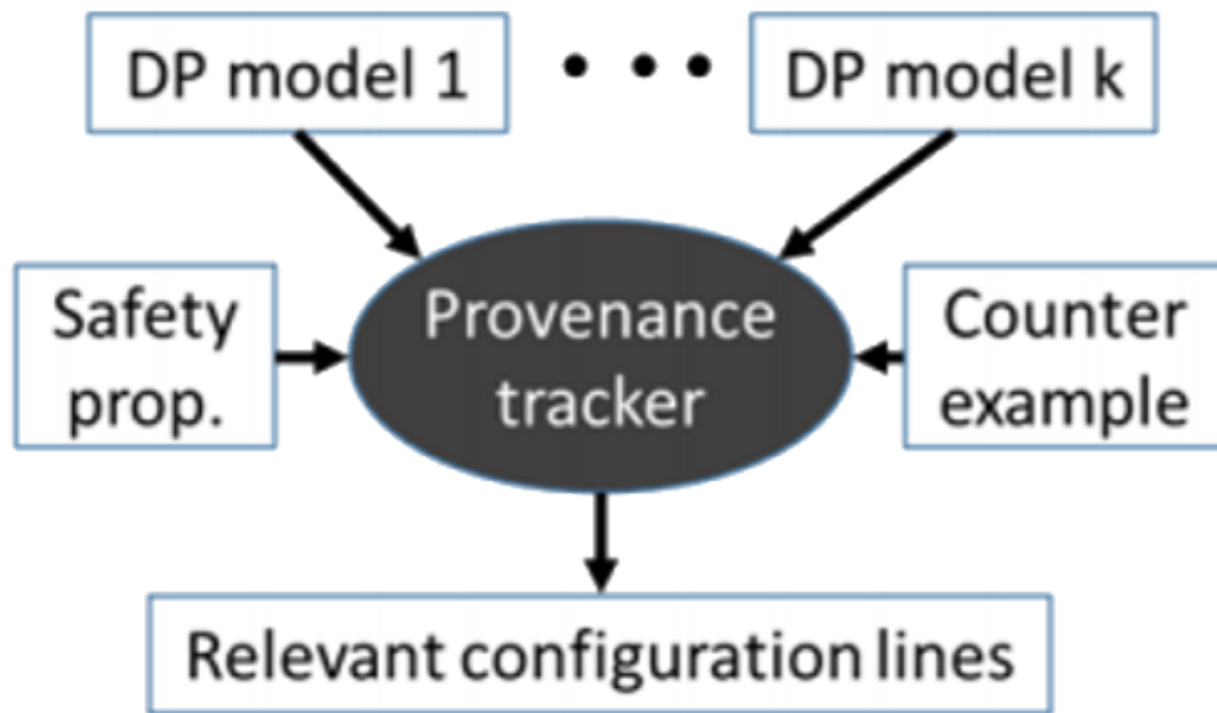
Stage 3: Data plane analysis



Counterexample of multipath consistency

```
{  
    IngressNode=n1,  
    SrcIp=0.0.0.0,  
    DstIp=10.0.0.2,  
    IpProtocol=0  
}
```

Stage 4: Report Provenance



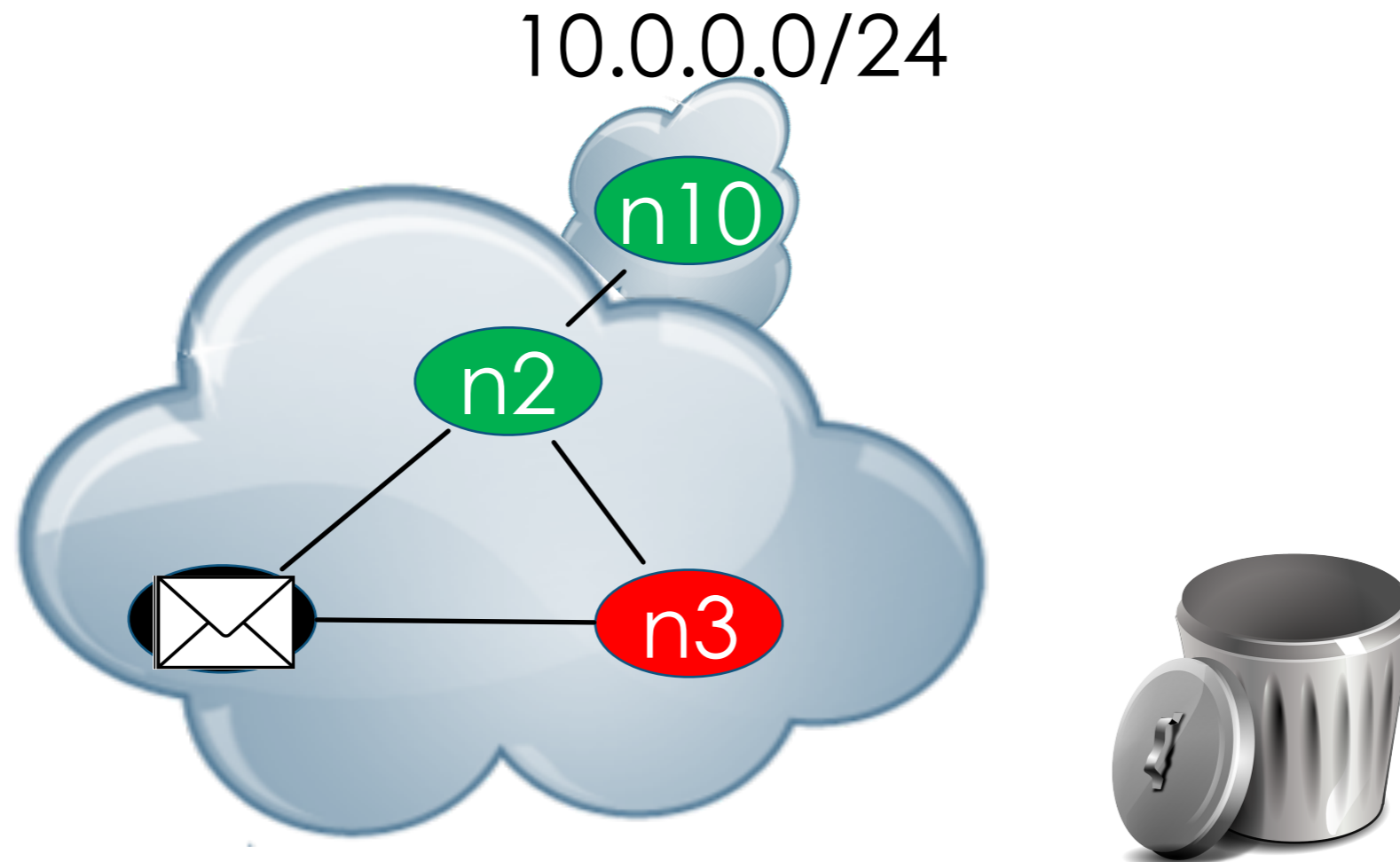
Counterexample packet traces

```
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

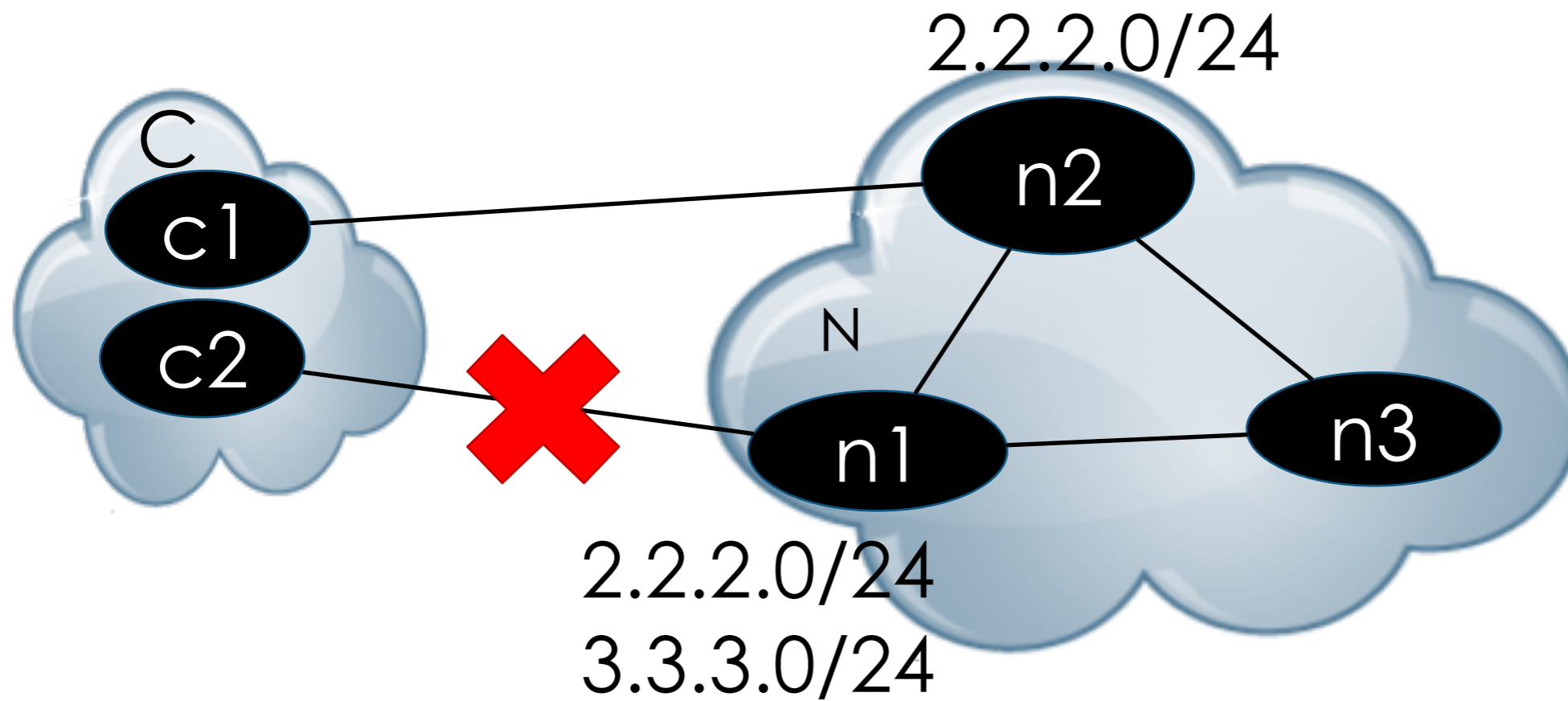
Multipath – disposition consistent on all paths



New Consistency Properties

Multipath – disposition consistent on all paths

Failure – reachability unaffected by failure

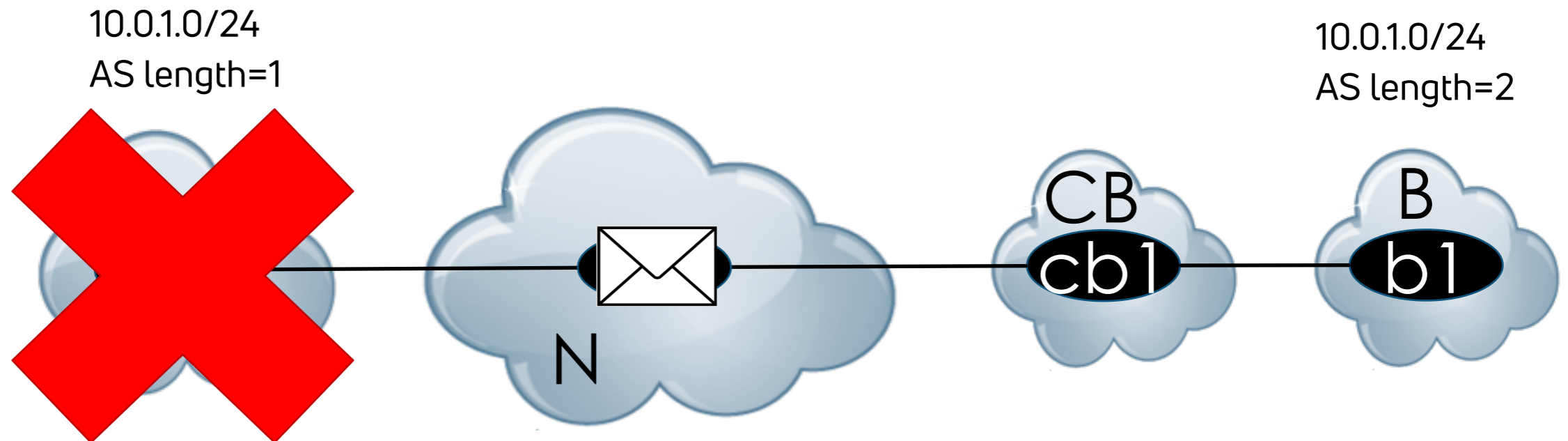


New Consistency Properties

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

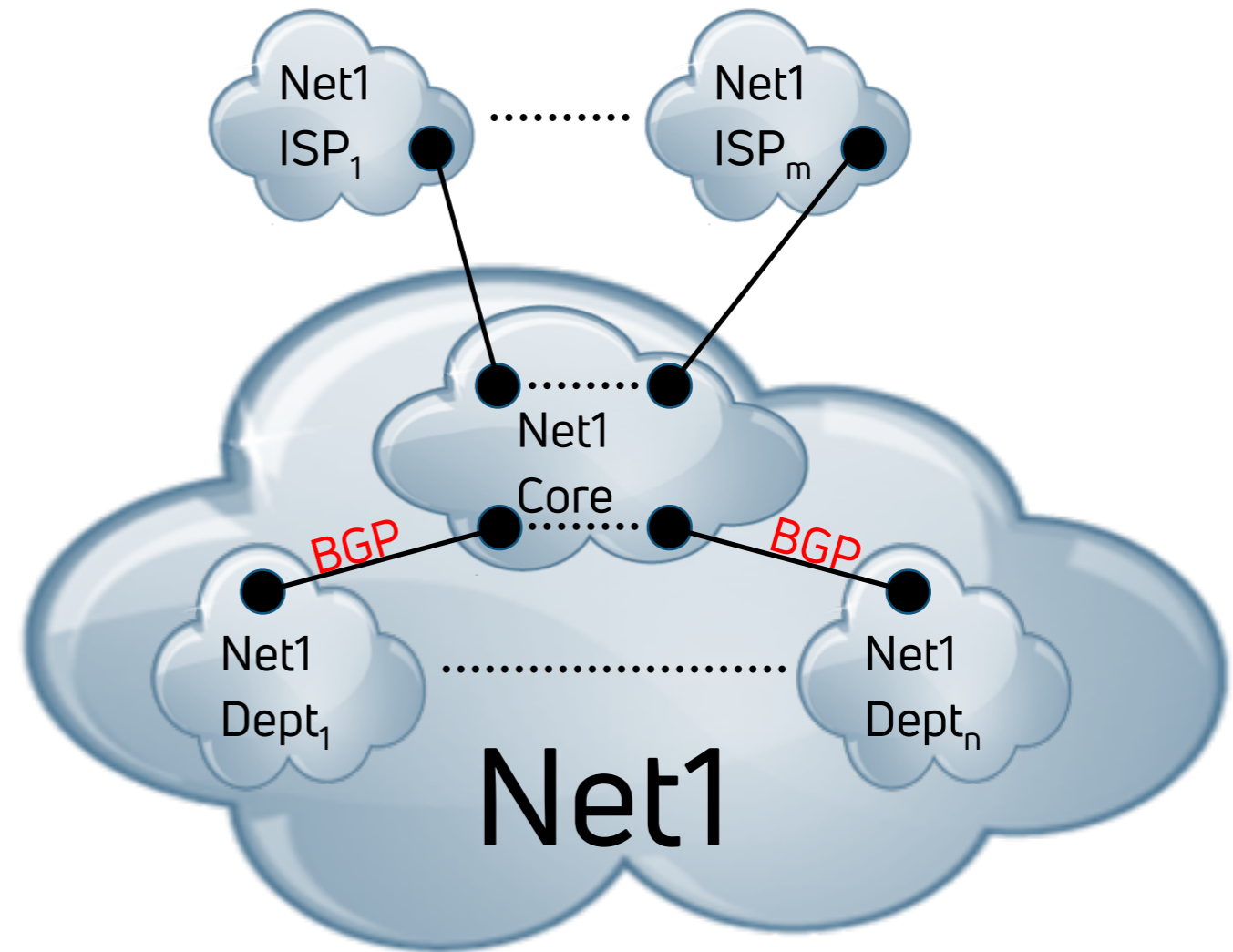
Two large university networks

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

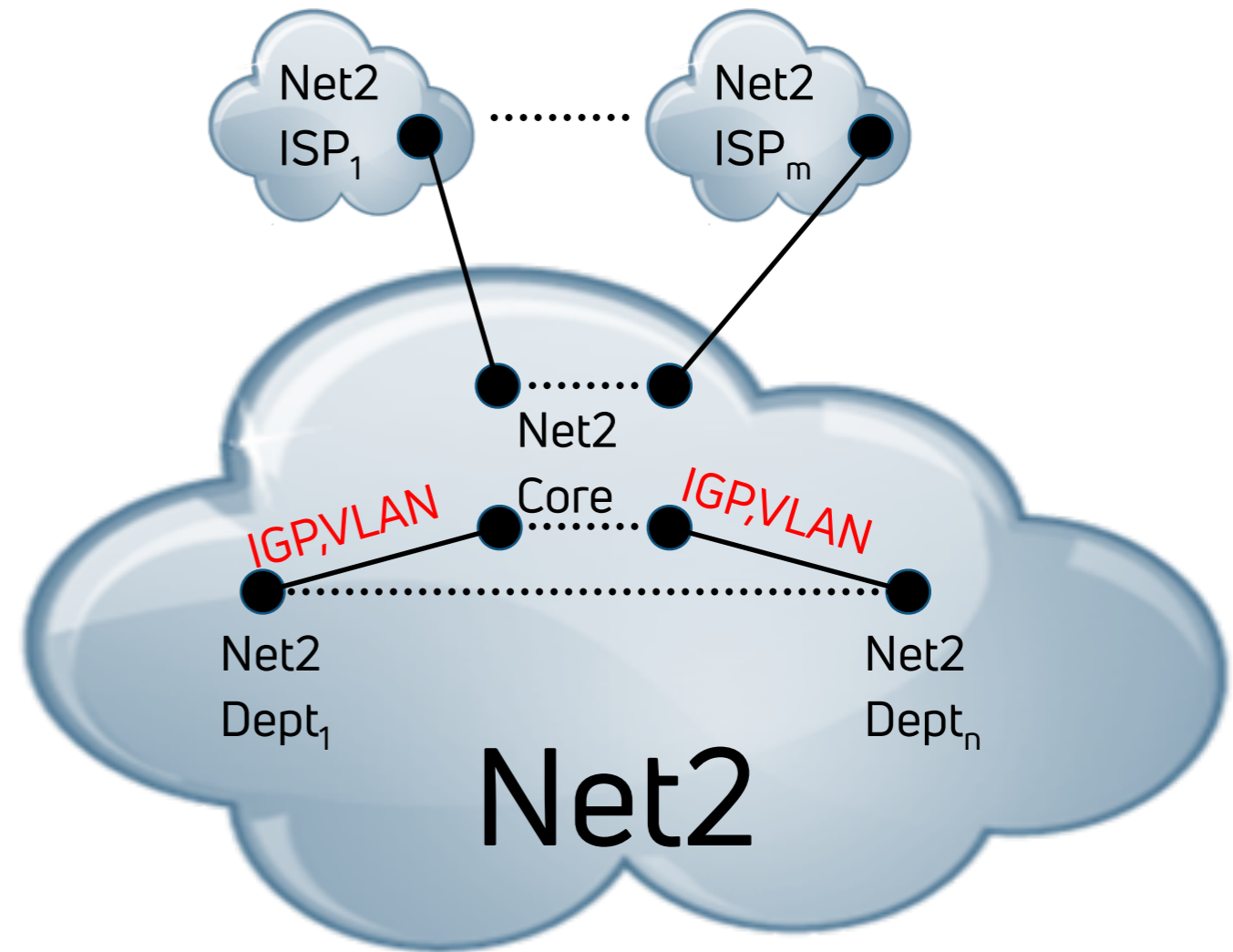
Two large university networks

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

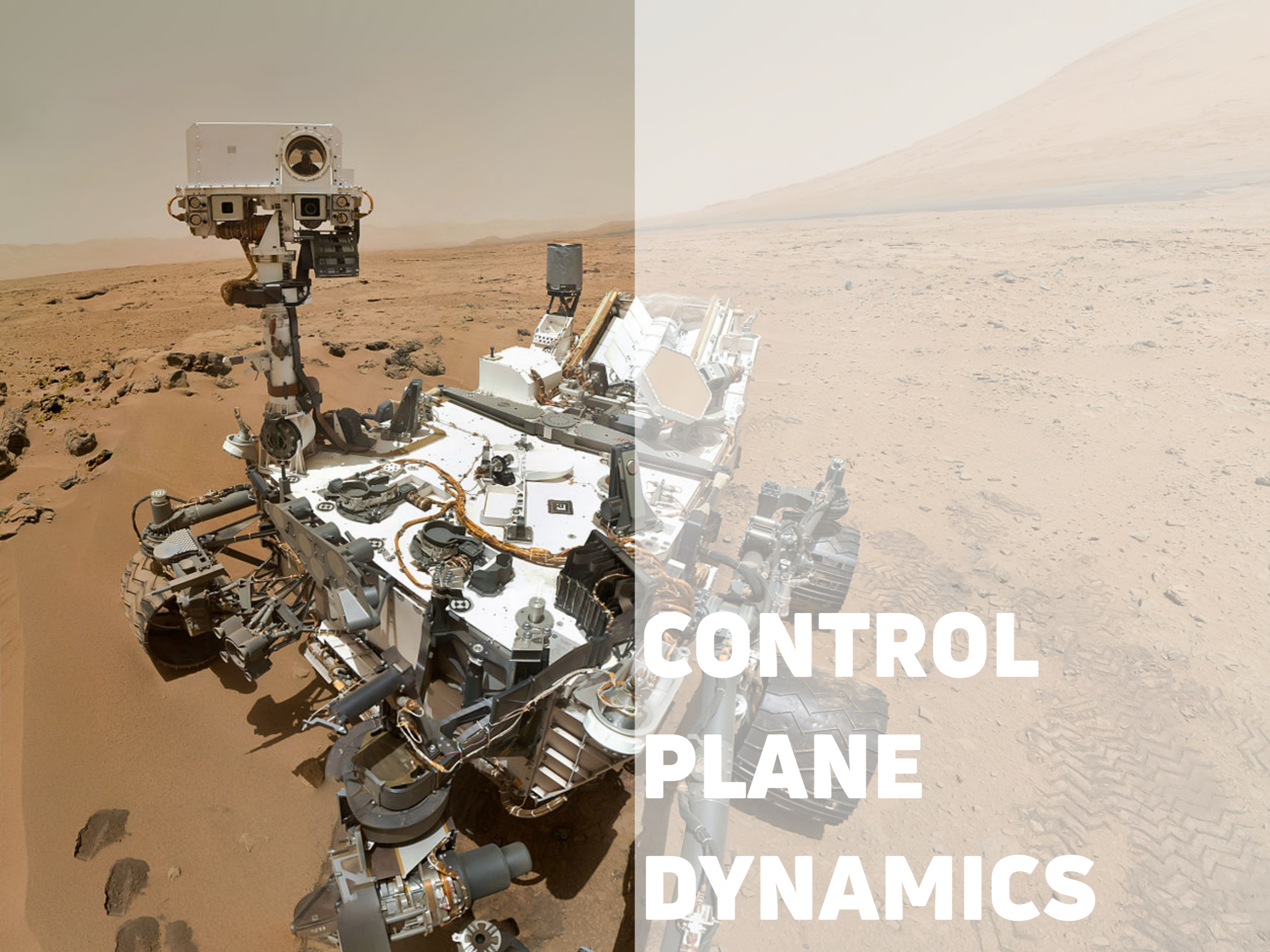


Results

	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)

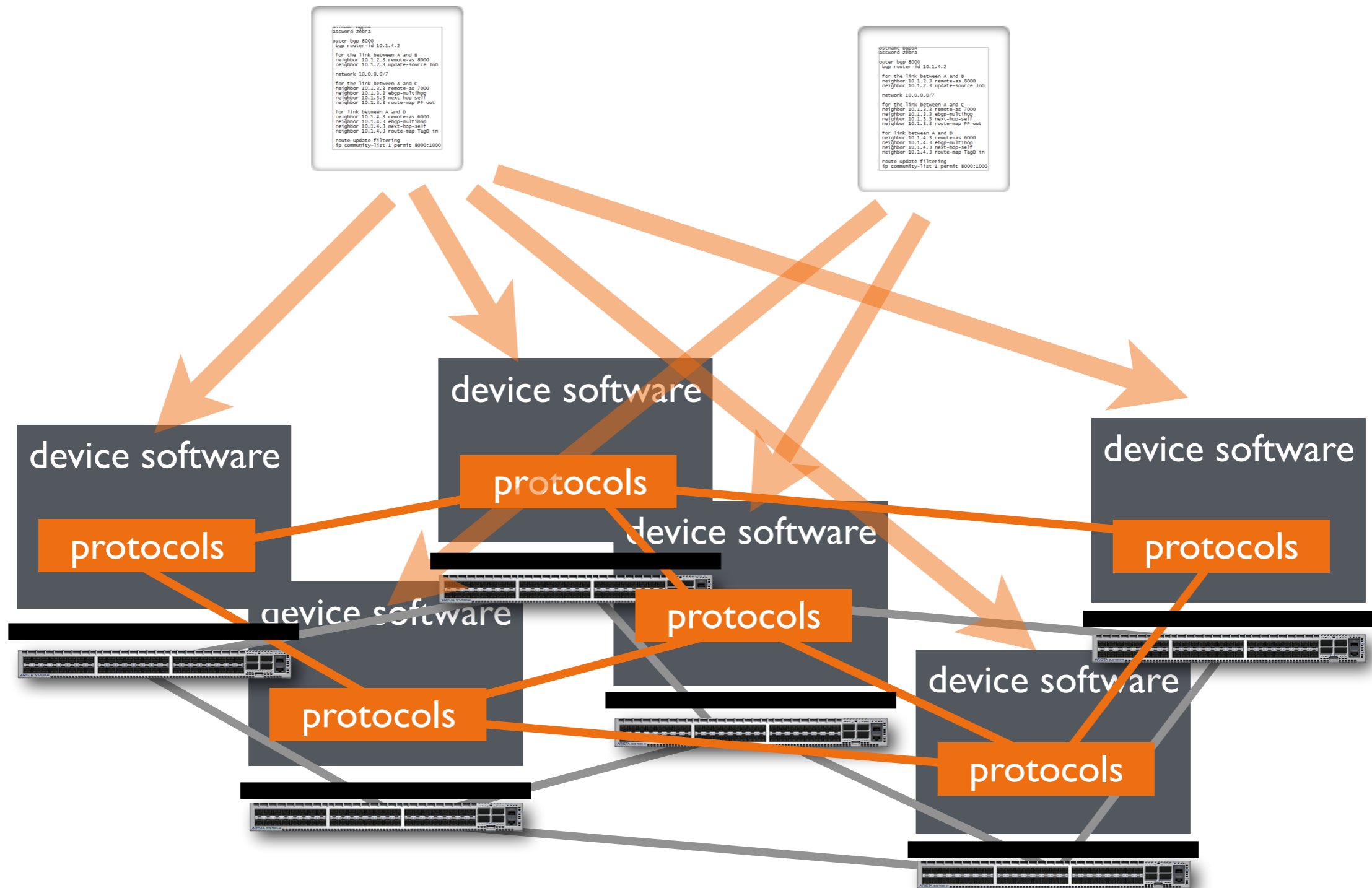


**CONTROL
PLANE
DYNAMICS**

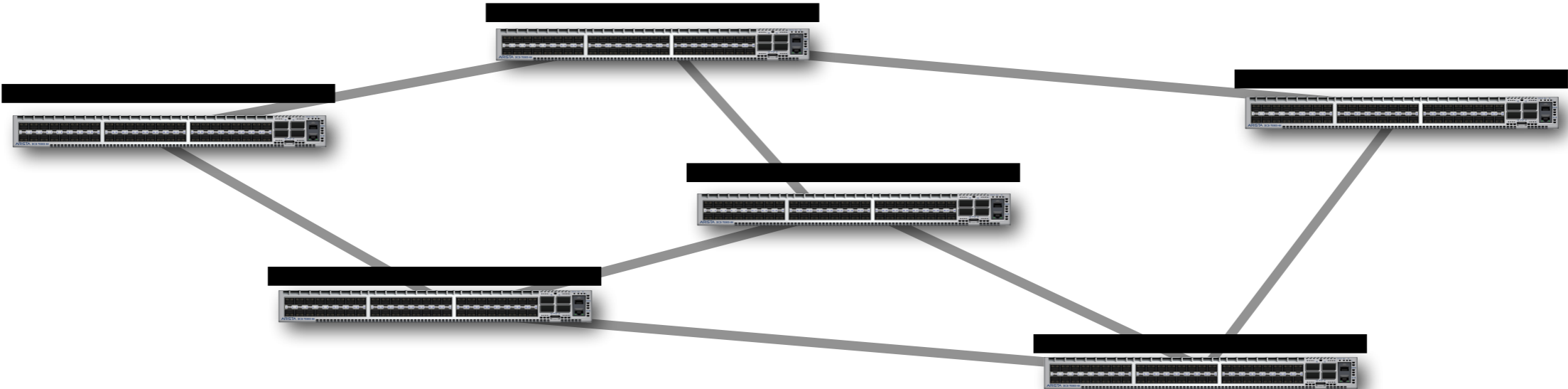
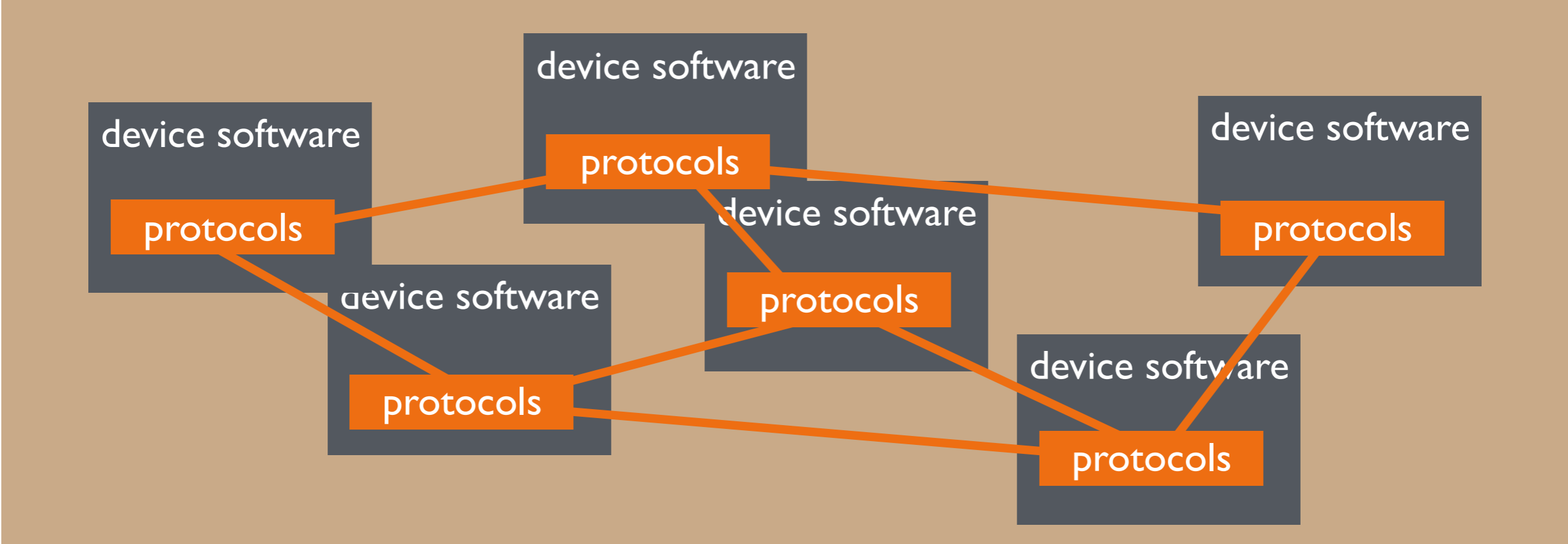
Landscape of approaches

	Data Plane Verification	Network Emulation	Control plane simulation	Configuration Verification
Configuration	Production run	Input	Input	Input
Control software	Emulated	Simulate a model run	Explore all model runs	
Data plane state	Input / Verified	Verified	Verified	Verified
Packet processing	Abstracted	Abstracted	Abstracted	Abstracted

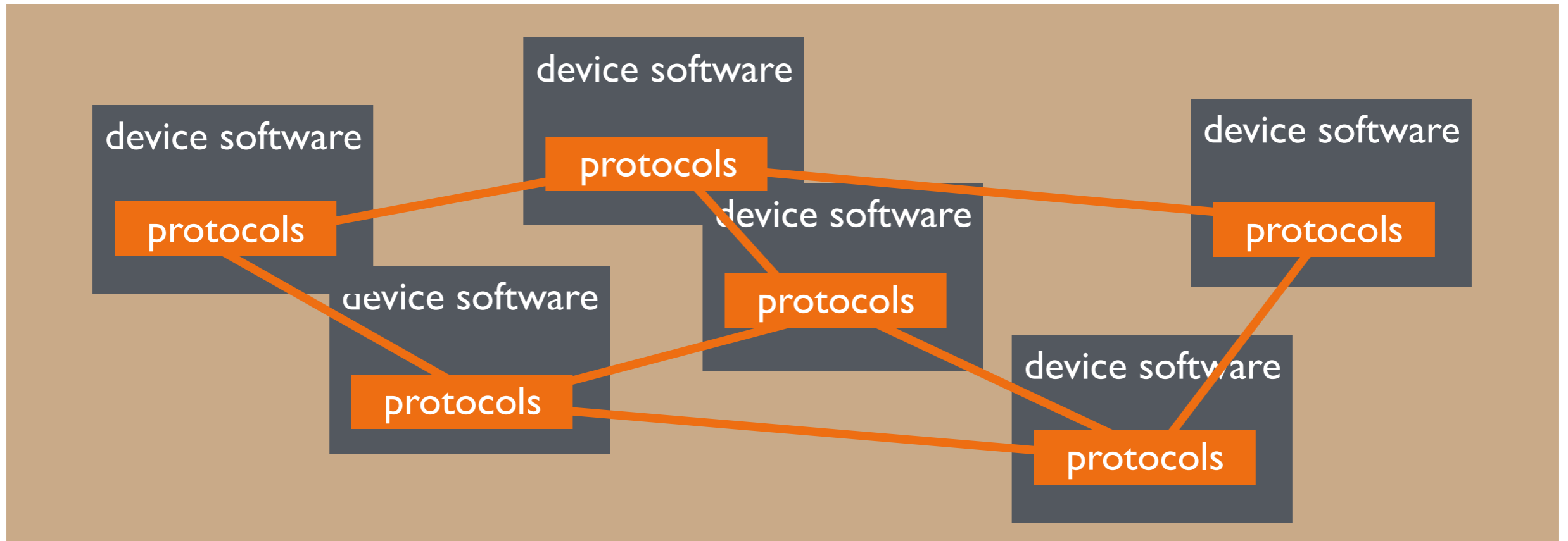
The Control Plane System



The Control Plane System



The Control Plane System



⚠ Failures and Failure Recovery

⚠ Non-determinism (Races etc)

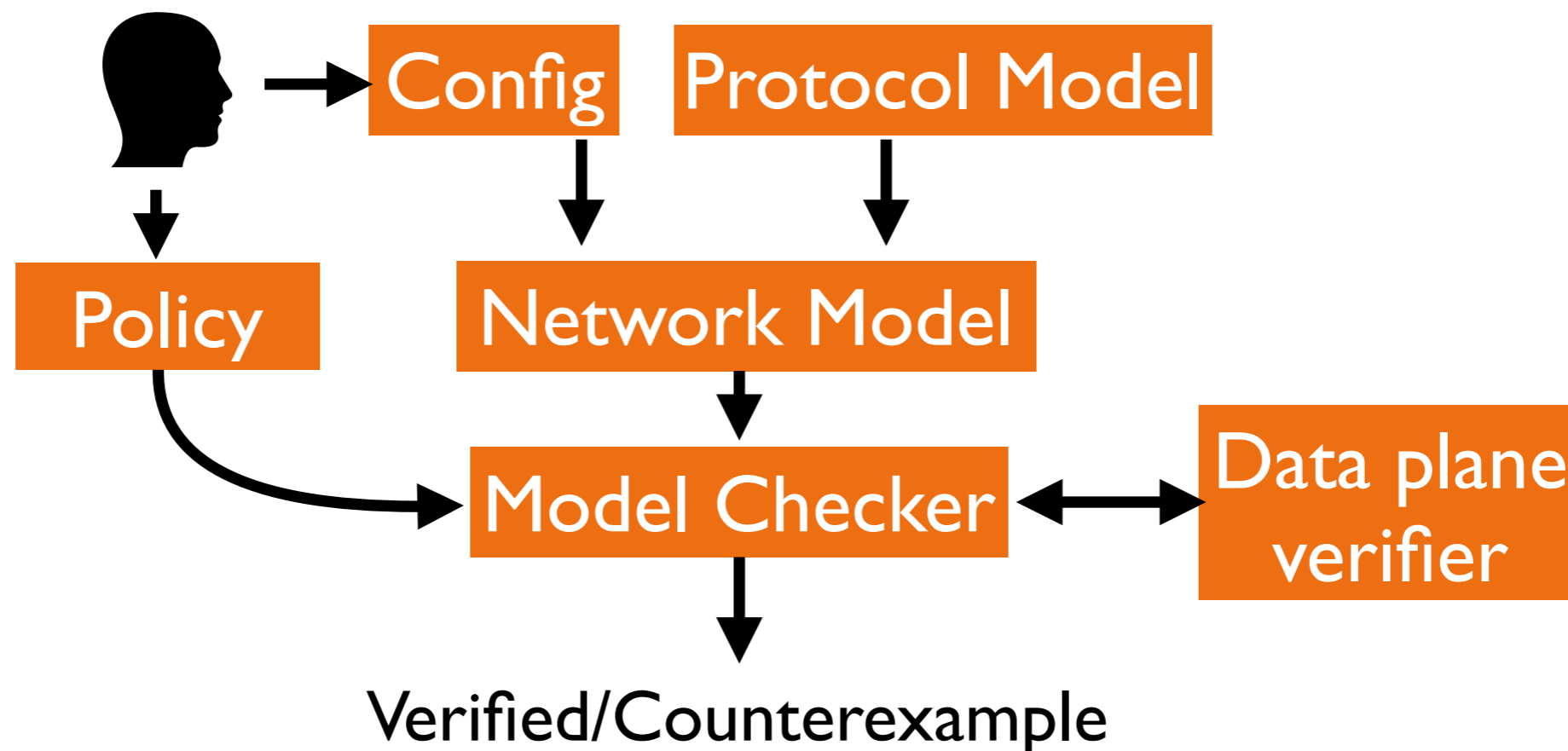
⚠ External input (Other N/Ws)

⚠ Turing Complete

Plankton: Verifying the Control Plane System

- Per - Equivalence Class modeling
- Control Plane + Environment \longrightarrow Program
- Explicit-state model checker to explore the network program
- Data plane verifier to evaluate predicates over the data plane states generated
- Specify temporal properties in the model checker over these predicates

Predicting Network Futures
With Plankton
Kheradmand, Prabhu, Godfrey,
Caesar, APNet 17



Optimization

Per-EC modeling

Reduced scope (Can't check *Packets A and B behave identically*). Better scalability.

Partial Order Reduction

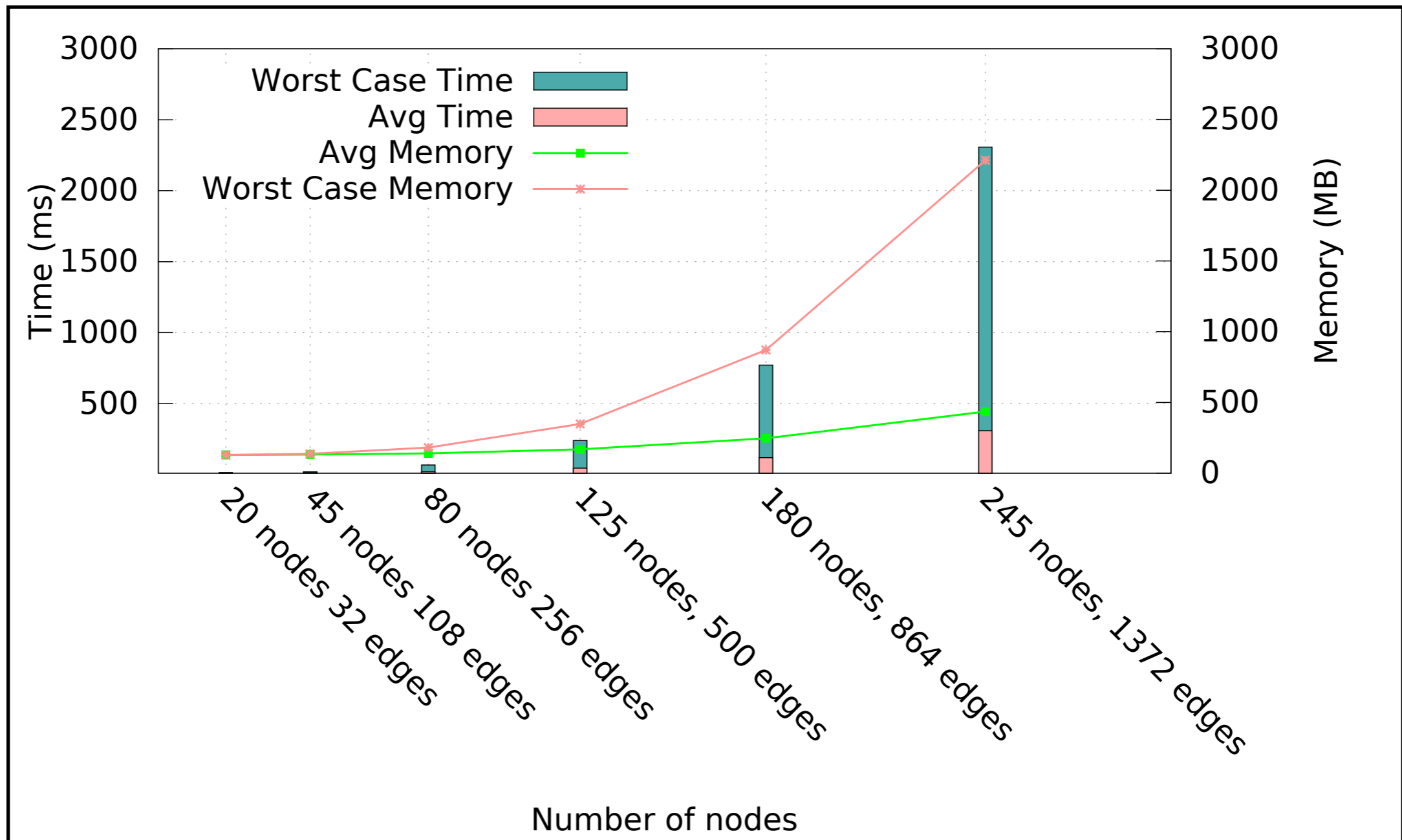
Eliminate paths of protocol convergence leading to the same final data plane.

Cone of Influence

Consider only environment actions that may cause intent violation.

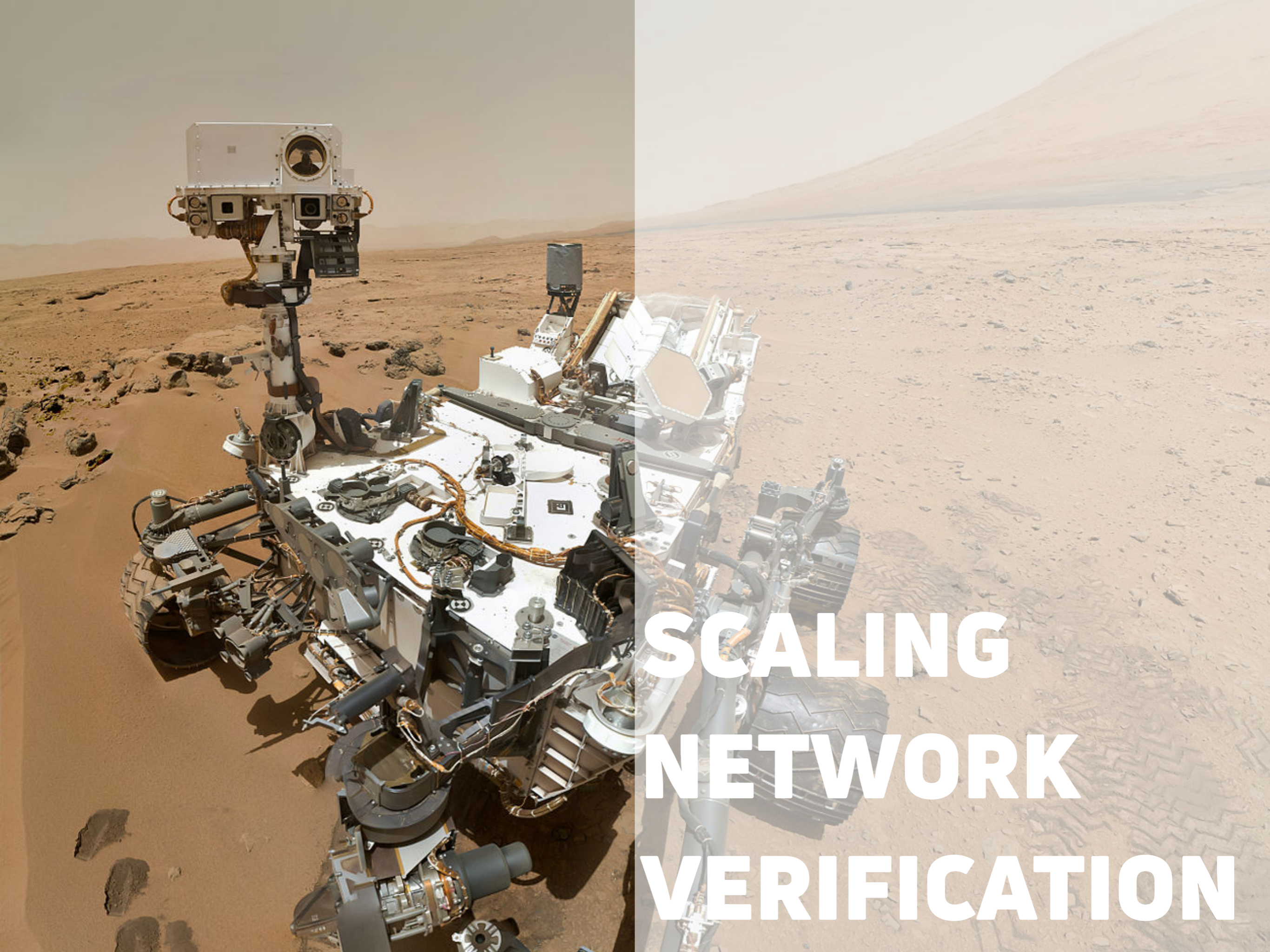
Some Numbers

- Data centers running BGP
- Paths from A to B should pass through C



Comparison of Config Analysis

Challenge	Simulation	Emulation	Verification
Catch vendor-specific issues	✗	✓	✗
Proprietary protocols	✗	✓	✗
Integrate with real devices	✗	✓	✗
New devices/ configuration	Hard	Easy	Hard
Resource Requirement	?	?	?
Explore all failures	Relatively Slow	Slow	Fast
Nondeterminism	✗	✗	✓
Control plane property verification	✗	✗	✓
Provenance	✓	✗	✓



**SCALING
NETWORK
VERIFICATION**

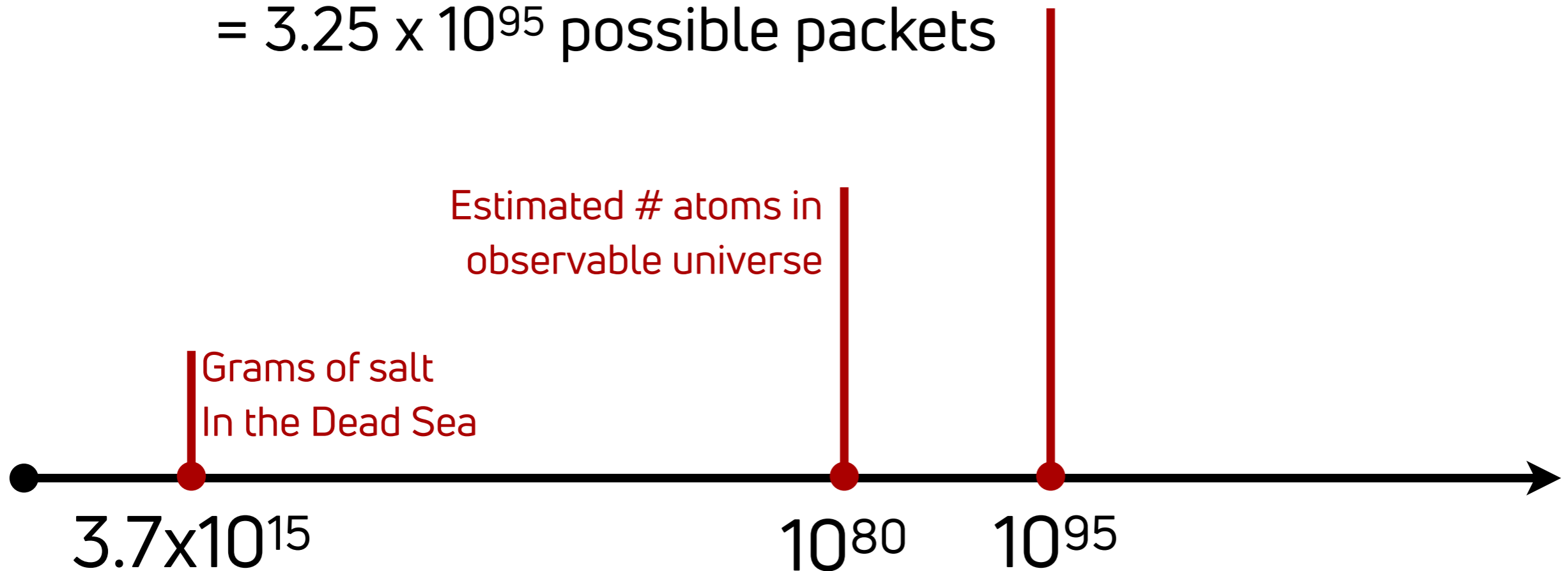
A little calculation...

theoretical packets

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

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

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



Protocols protocols everywhere...

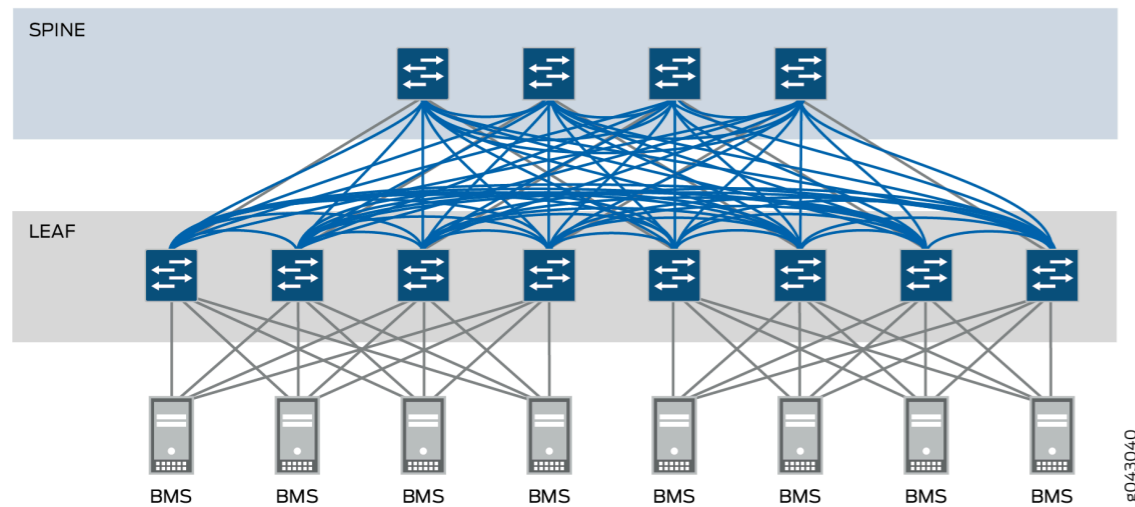
Typical Internet Data Frame



Abstraction (What to model)

DL	DST	-	48	bits
IP	DST	-	32	bits
IP	SRC	-	32	bits
TP	TYPE	-	8	bits
TP	SRC	-	16	bits
TP	DST	-	16	bits

Protocols protocols everywhere...



VXLAN - Commonly used
in datacenters

Image: Juniper Networks

DL	DST	-	48	bits	x	2
IP	DST	-	32	bits	x	2
IP	SRC	-	32	bits	x	2
TP	TYPE	-	8	bits		
TP	SRC	-	16	bits	x	2
TP	DST	-	16	bits	x	2
VXLAN		-	24	bits		

Analogous to
state explosion

Equivalence Classes!

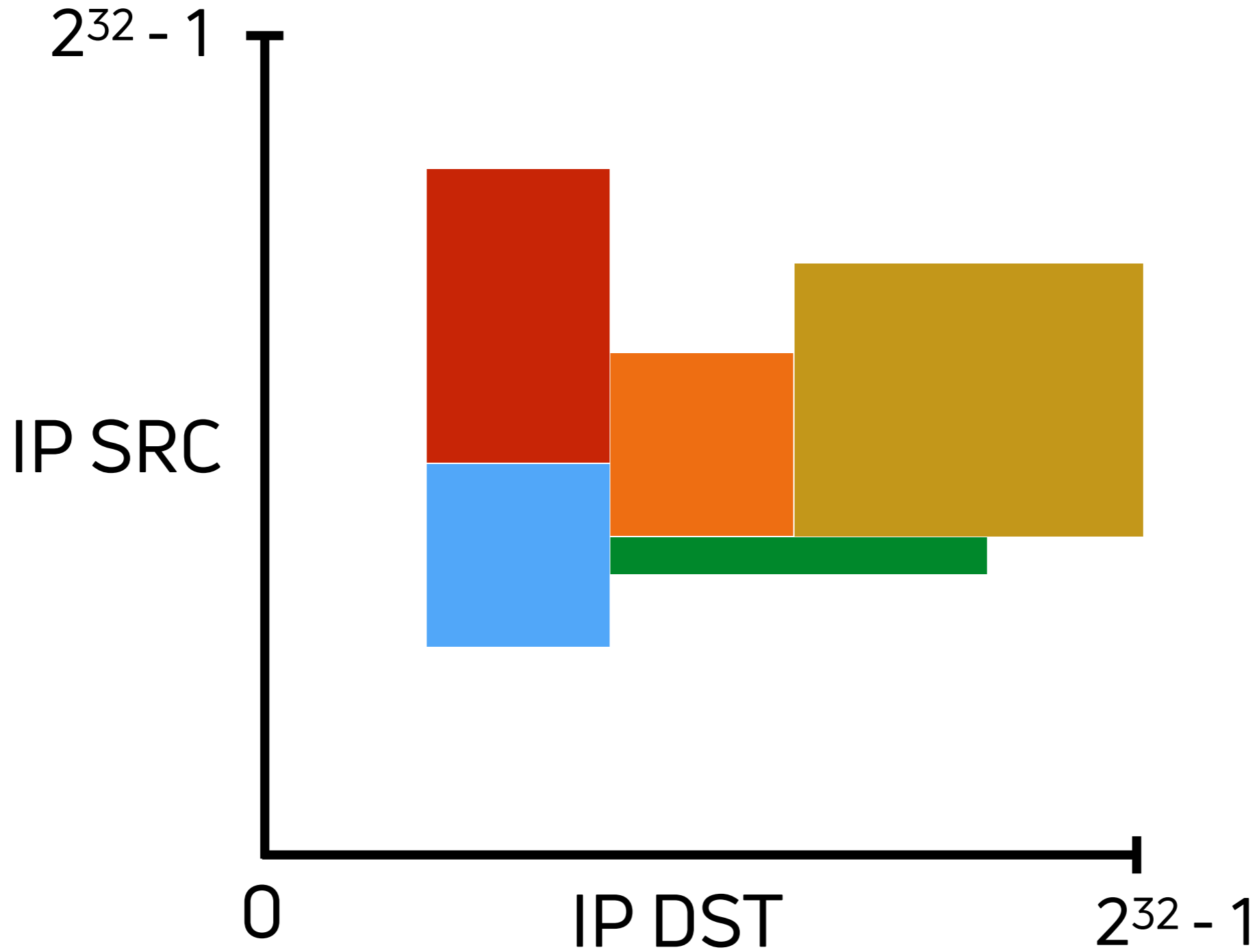
Choice of data structures

How can an packet
equivalence classes be stored
and manipulated **compactly**?

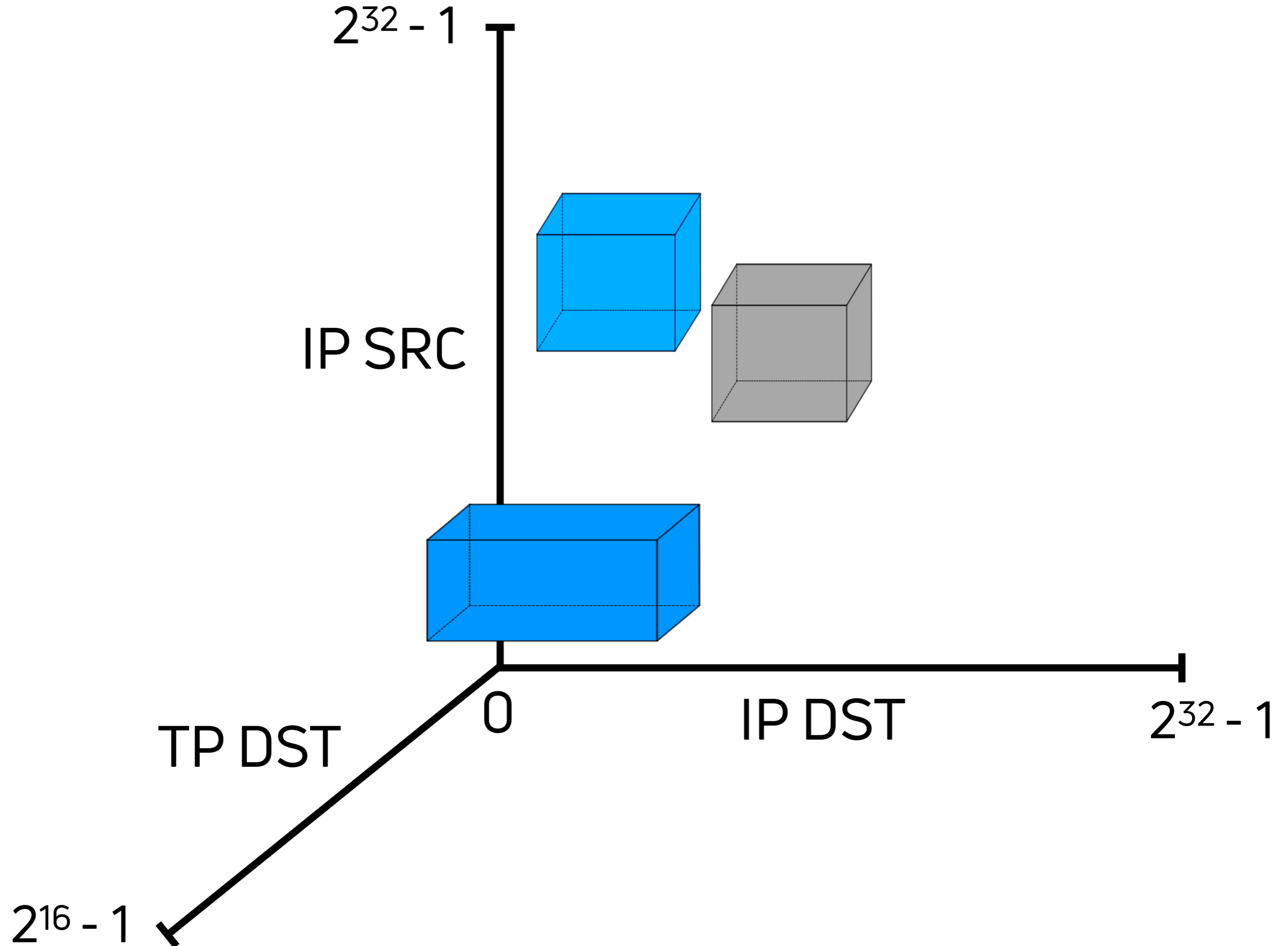
Choice of Data Structures - Ranges



Choice of Data Structures - Ranges



Choice of Data Structures - Ranges



Choice of Data Structures - Bit Vectors

Rule 1: 10110***

Rule 2: 10000***

EC 1: 10110***

EC 2: 10000***

EC 3: 0***** U

11***** U

1010*** ...

Header Space Analysis:
Static Checking for Networks
Kazemian, Varghese and
McKeown
NSDI 2012

Set Difference is Expensive! So don't evaluate

Rule 1: 10110***

Rule 2: 10000***

EC 1: 10110***

EC 2: 10000***

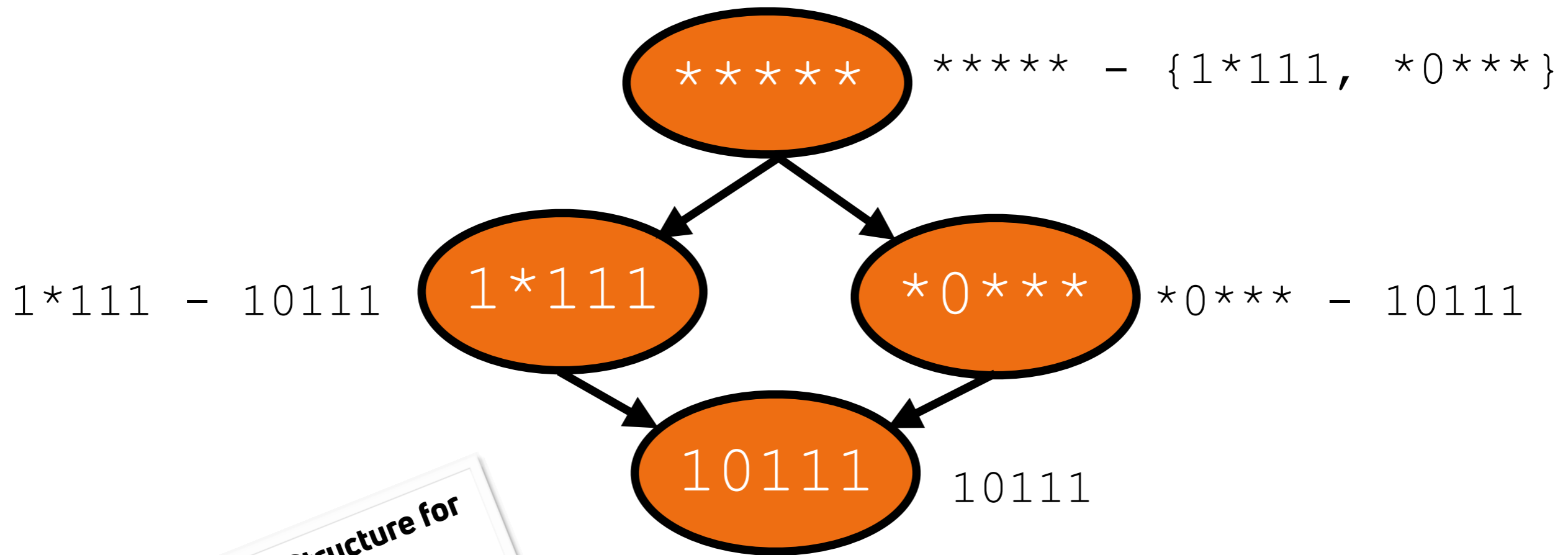
EC 3: ***** \

(10110*** U

10000***)

**Checking Beliefs in Dynamic
Networks**
Lopes, Bjørner,
Godefroid, Jayaraman and
Varghese
NSDI 2015

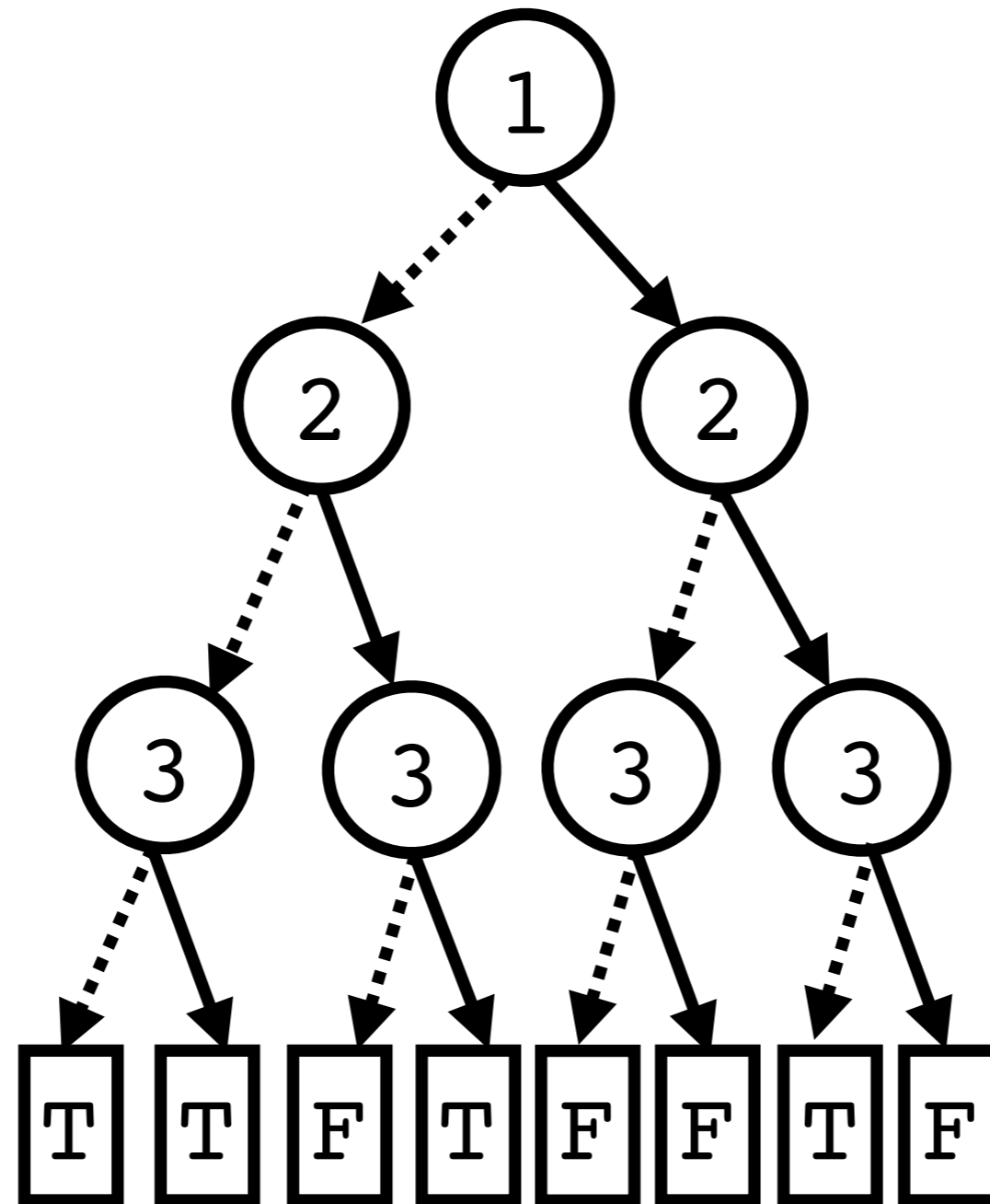
Choice of Data Structures - DDNFs



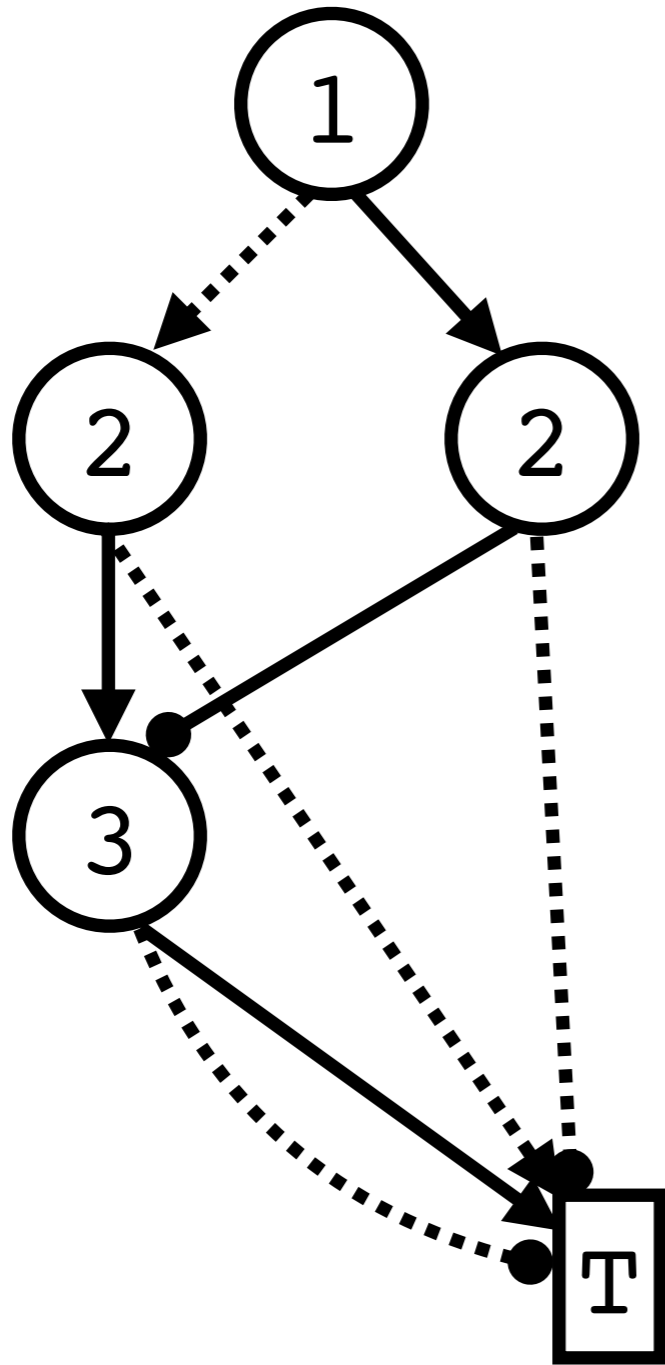
ddNF: An Efficient Data Structure for Header Spaces

Bjorner, Juniwal, Seshia, Varghese,
Mahajan
HVC 2016

Choice of Data Structures - BDDs



Choice of Data Structures - BDDs



Designed to mitigate the state explosion problem in model checking

Allows modeling packet modification etc

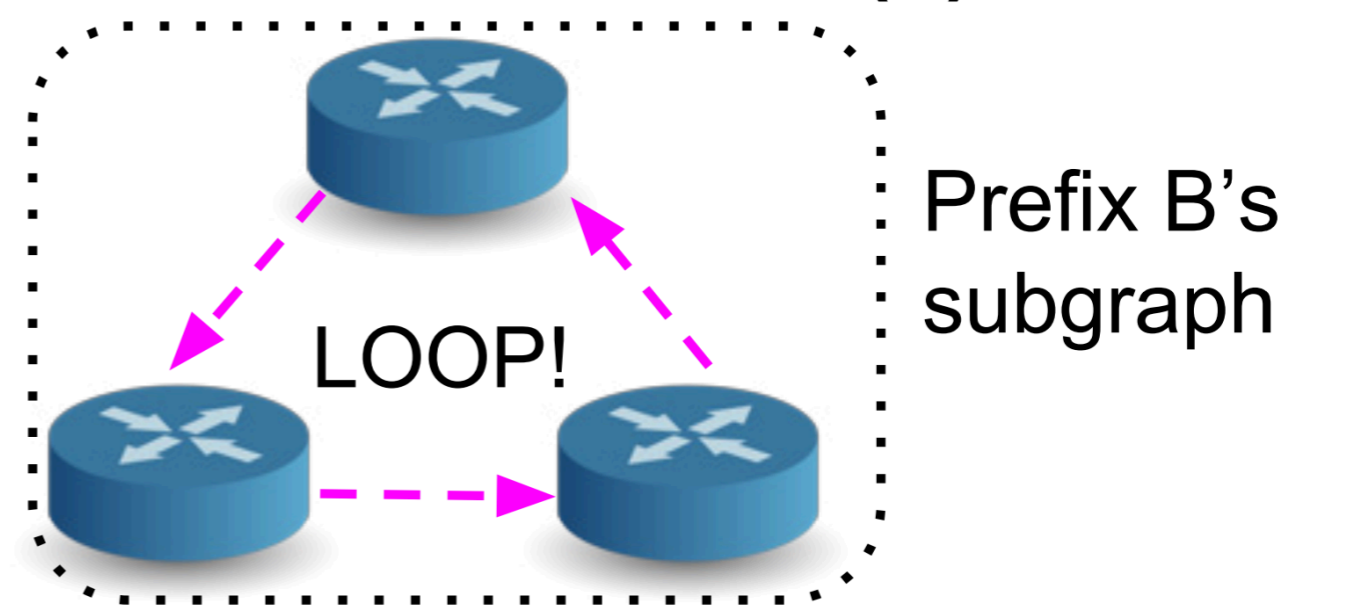
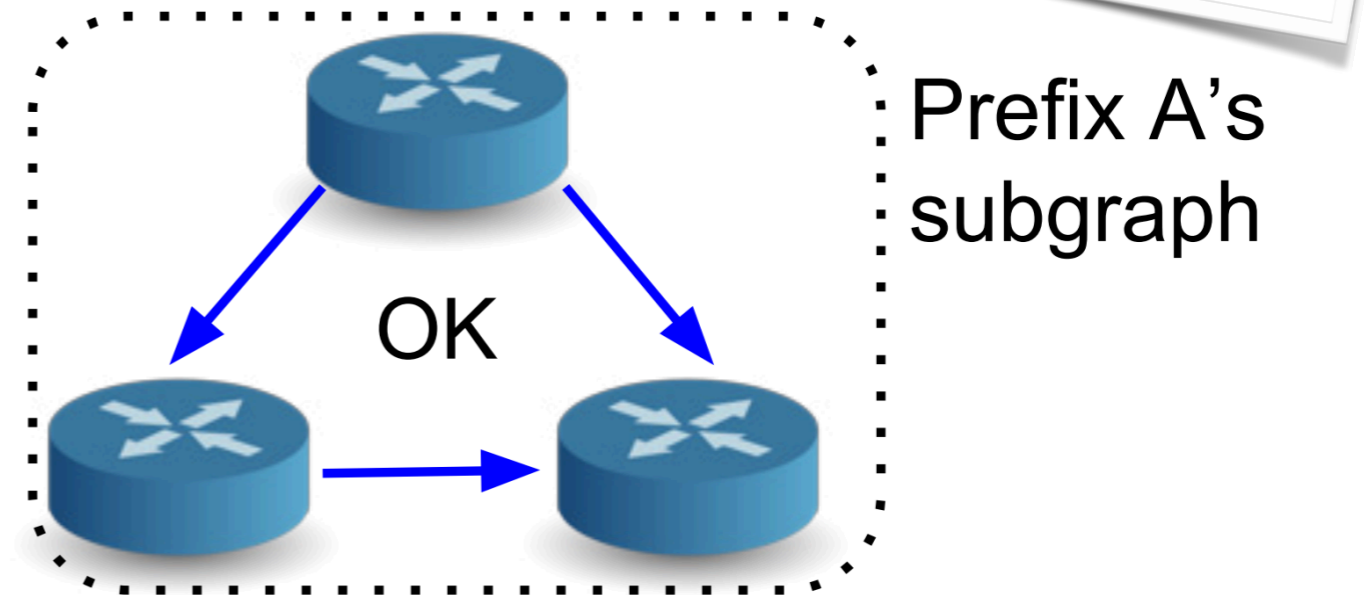
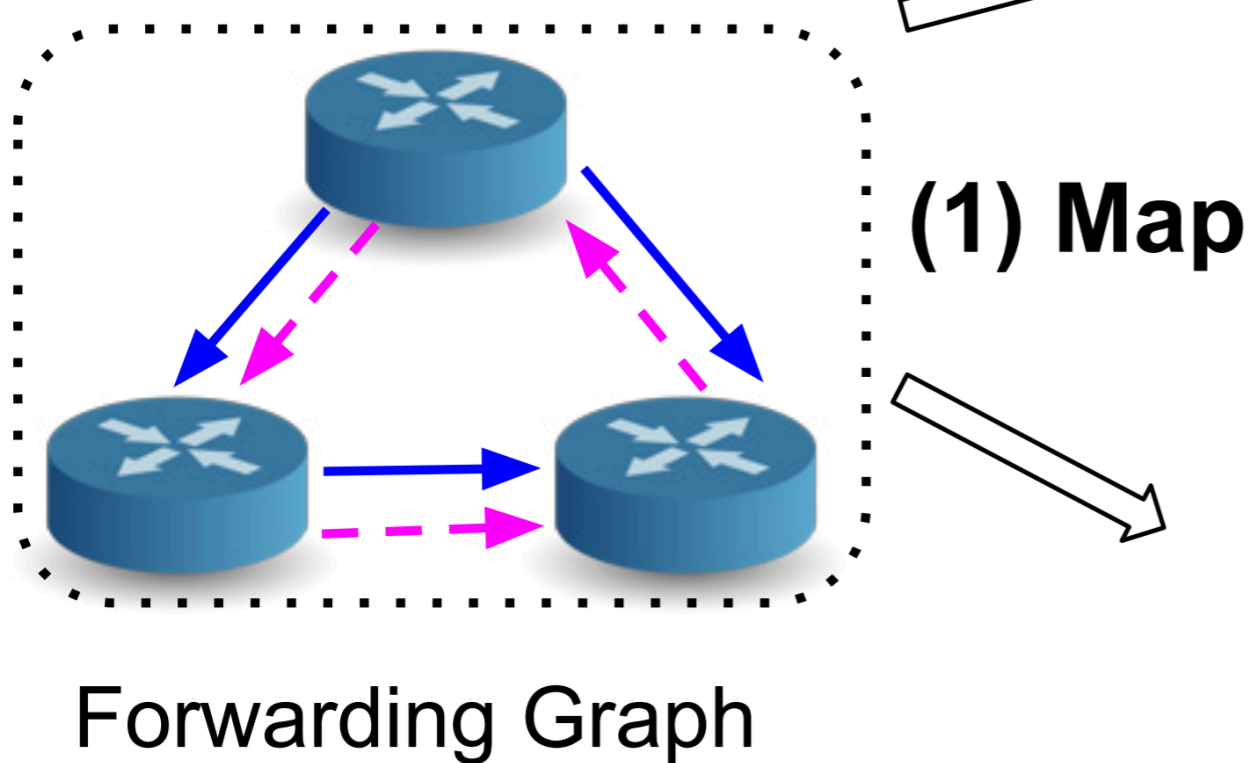
High overhead per bit. But useful for relatively large and diverse networks.

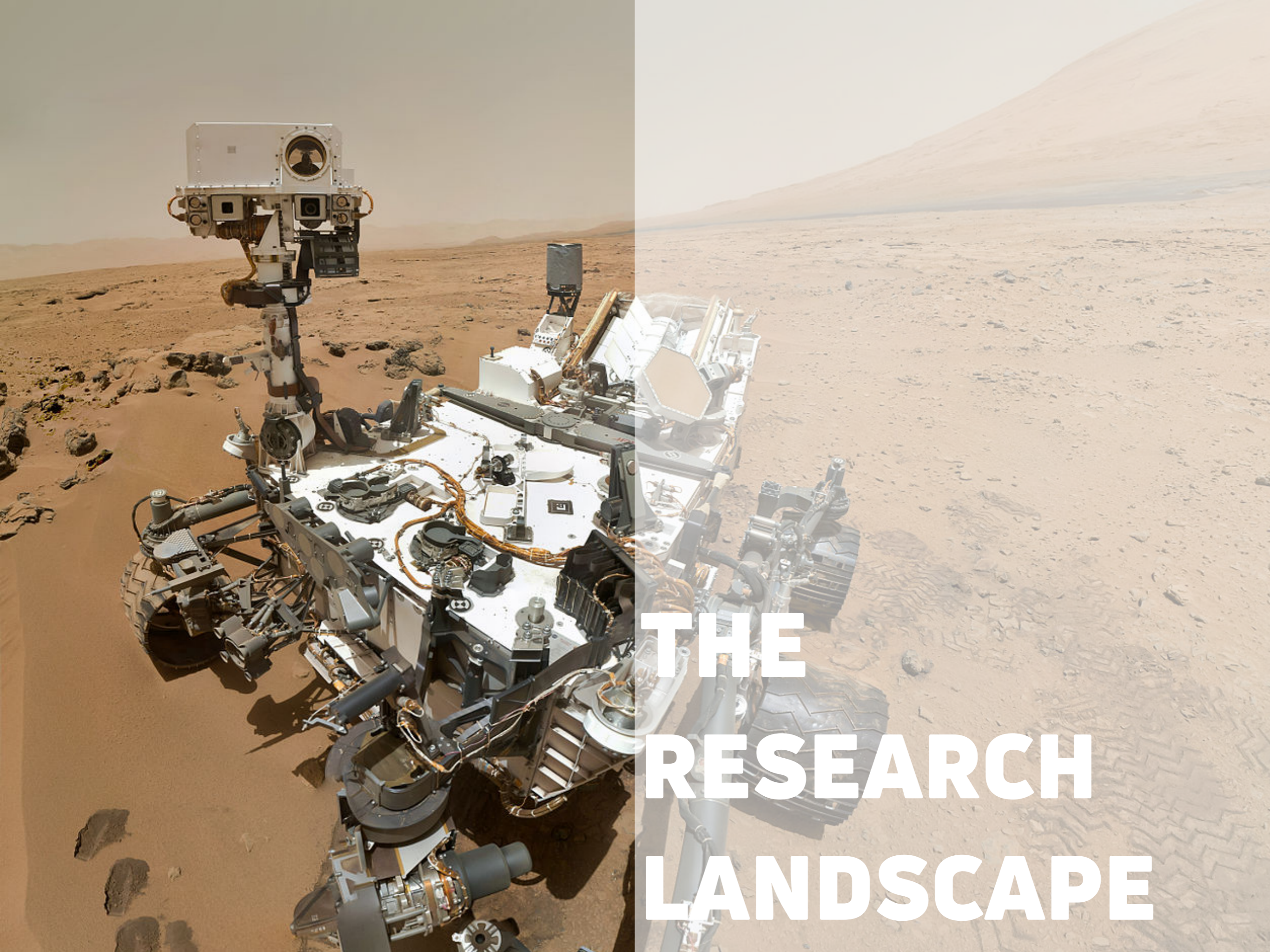
Divide and Conquer

Verification as Map-Reduce

Libra: Divide and Conquer to Verify Forwarding Tables in Huge Networks
Zeng, Zhang, Ye, Jeyakumar, Ju, Liu, McKeown, Vahadat
NSDI 2014

- Prefix A
- - - Prefix B
- Process Boundary

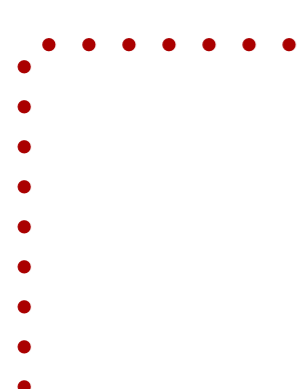




**THE
RESEARCH
LANDSCAPE**

Taxonomy: Abstractions of the Network

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



Configuration

Configuration verification

Control software

Controller verification & verifiable control languages

Data plane state

Data plane verification

Packet processing

Software switch verification, Formal P4 semantics

Data plane verification

Configuration

Control
software

Data plane
state

Packet
processing

Static

- On static reachability in IP networks [Xie, Zhan, Maltz, Zhang, Greenberg, Hjalmtysson, Rexford, INFOCOM '05]
 - 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]
- DeltaNet [Horn, Kheradmand, Prasad] NSDI'17]

Data plane verification (cont'd)

Configuration

Control
software

Data plane
state

Packet
processing

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]

Richer data plane models

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

Control software verification

Configuration

Control
software

Data plane
state

Packet
processing

Verifiable controllers & control languages

- NICE [Canini, Venzano, Perešini, 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]

Configuration verification

Configuration

Control
software

Data plane
state

Packet
processing

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]
- ARC [Gember-Jacobson, Viswanathan, Akella, Mahajan SIGCOMM'16]
- ERA [Fayaz, Sharma, Fogel, Mahajan, Millstein, Sekar, Varghese, OSDI'16]
- Minesweeper [Beckett, Gupta, Mahajan, Walker SIGCOMM'17]
- Plankton [Prabhu, Kheradmand, Godfrey, Caesar APNet'17]
- CrystalNet [Liu, Zhu, Padhye, Cao, Tallapragada, Lopes, Rybalchenko, Lu, Yuan SOSP'17]

Packet processing verification

Configuration

Control
software

Data plane
state

Packet
processing

Verification of data plane software and data plane languages

- Software Dataplane Verification [Dobrescu, Argyraki, NSDI'14]
- Executable Formal Semantic of P4 and Applications [Kheradmand, Rosu, P4 Workshop'17]
- A Formally Verified NAT [Zaostrovnykh, Pirelli, Pedrosa, Argyraki, Candea, SIGCOMM'17]

Packet processing verification

Configuration

Control
software

Data plane
state

Packet
processing

Click Modular
Router

Properties

Bounded Execution

Crash Freedom

Security - Filtering

Techniques:

Symbolic Execution

Modular
Decomposition

Software Dataplane Verificaiton
Dobrescu and Argyraki
NSDI 2014
Best Paper!

P4: Data plane programmability

Configuration

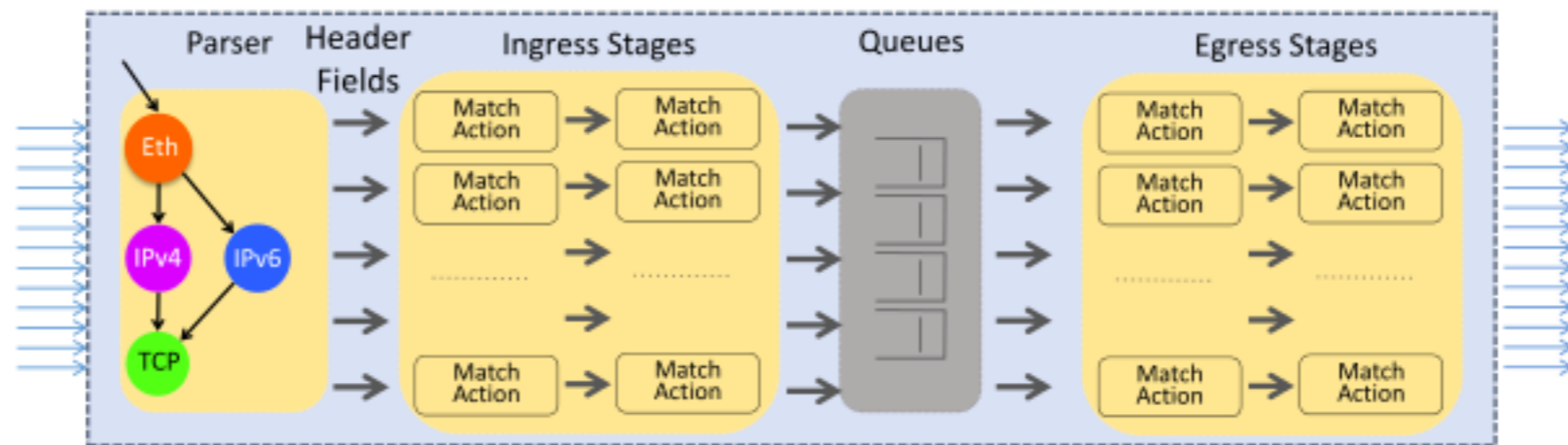
Control software

Data plane state

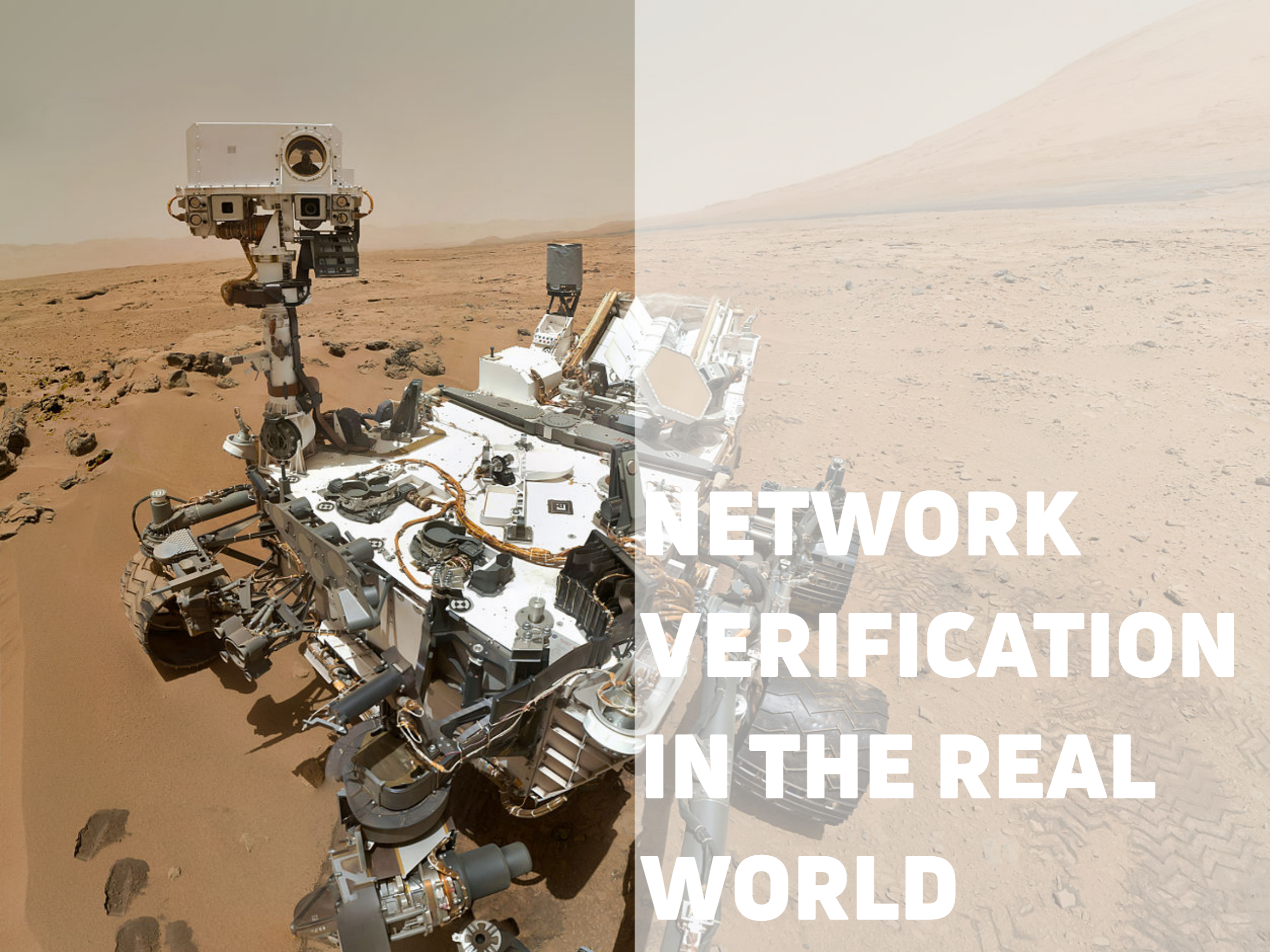
Packet processing

Programmable
Data Plane

- Support for new protocols
- Customize packet parsing, actions etc



P414 abstract switch model



**NETWORK
VERIFICATION
IN THE REAL
WORLD**

Industry efforts

Multiple startups pursuing general-purpose network verification for enterprises

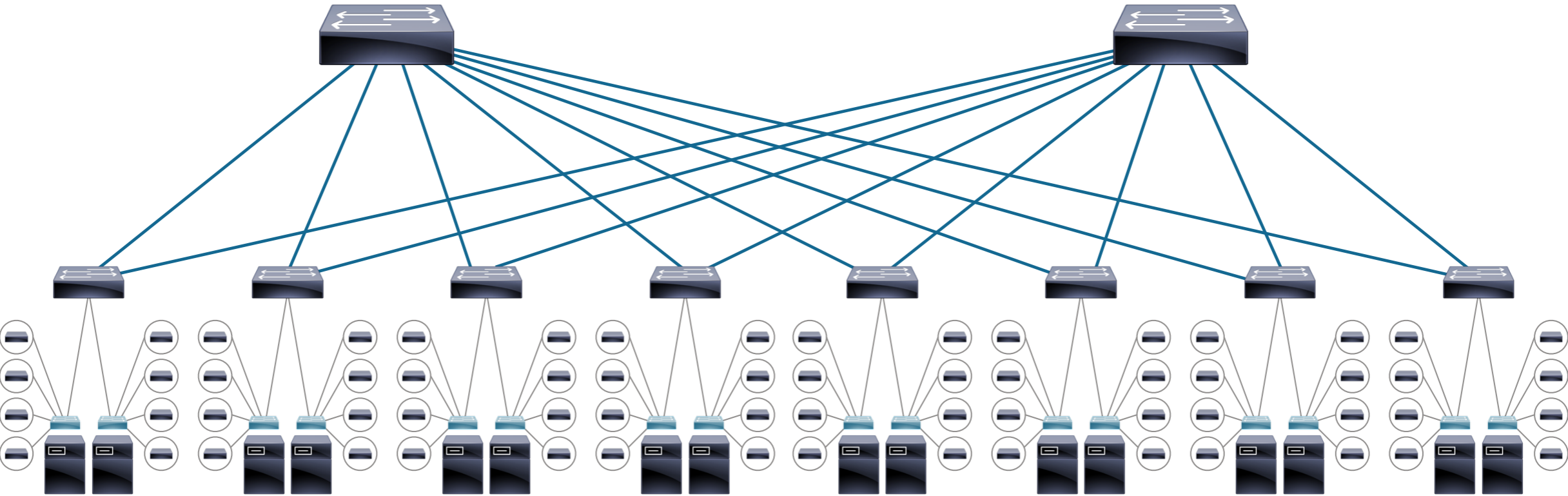
Special purpose efforts

- Microsoft, Google: Verification of internal infrastructure
- Amazon: Verification of virtual network access controls

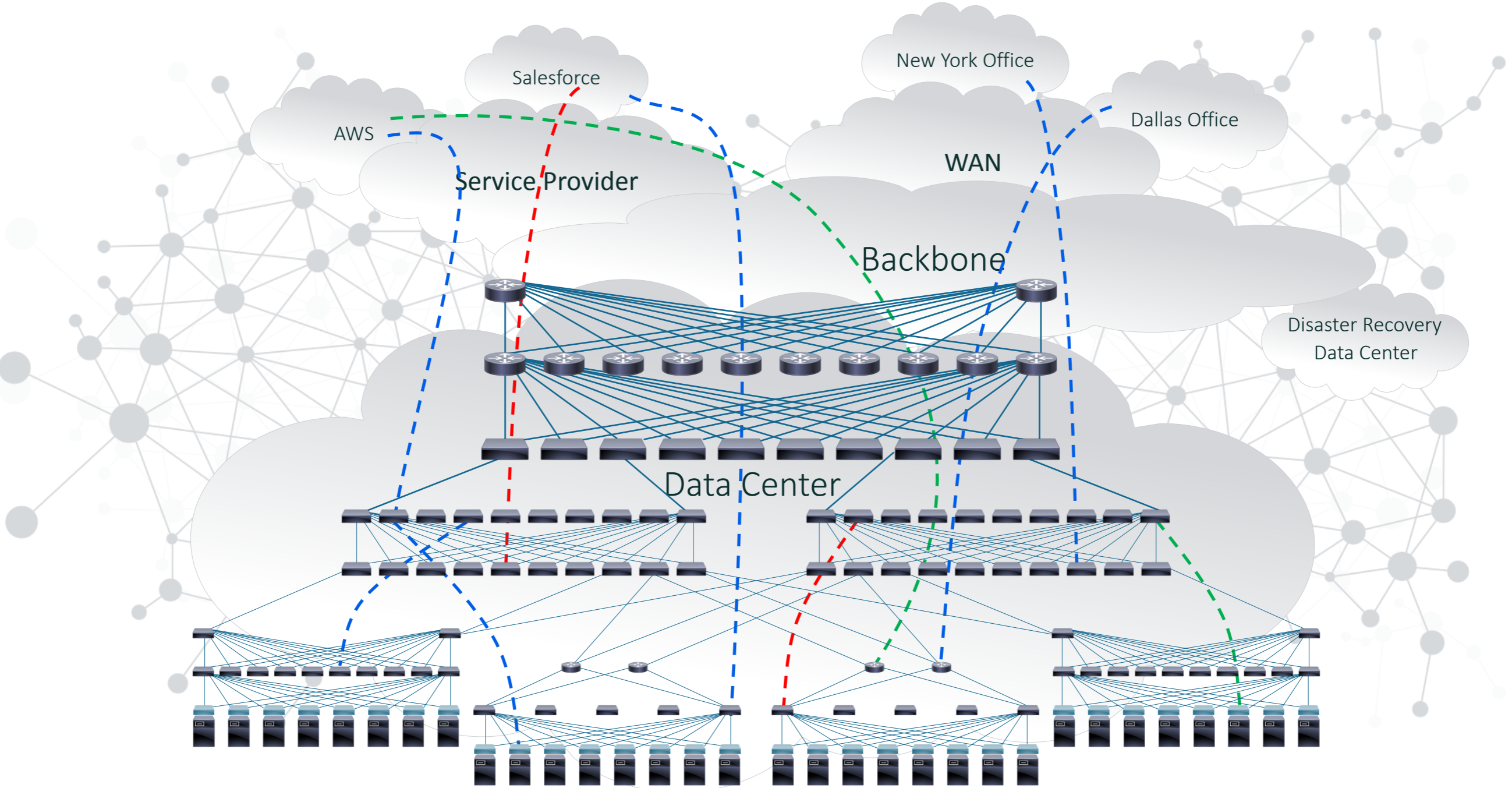
Gartner grouping verification in “intent-based networking” category

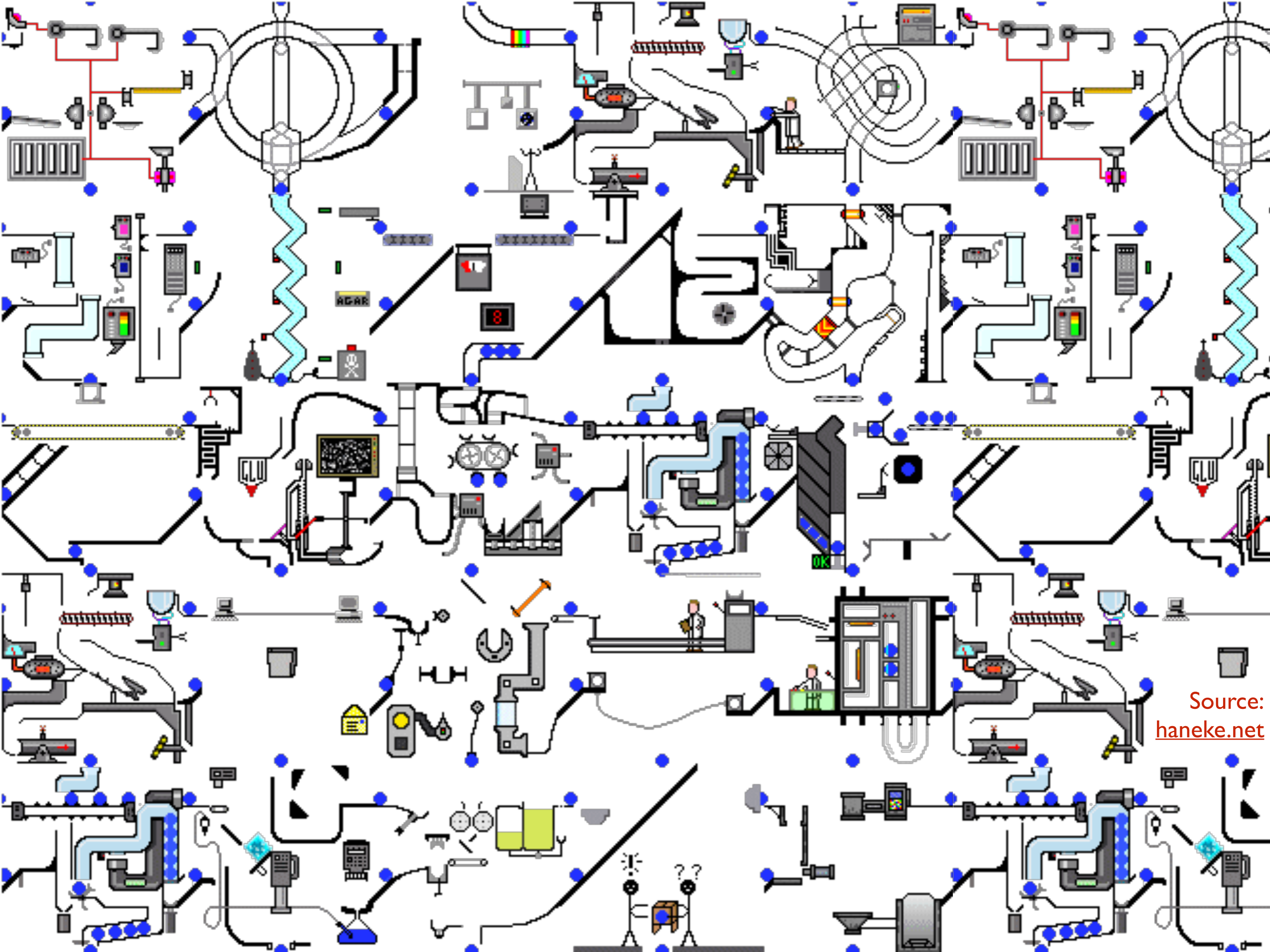
What have we learned?

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

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

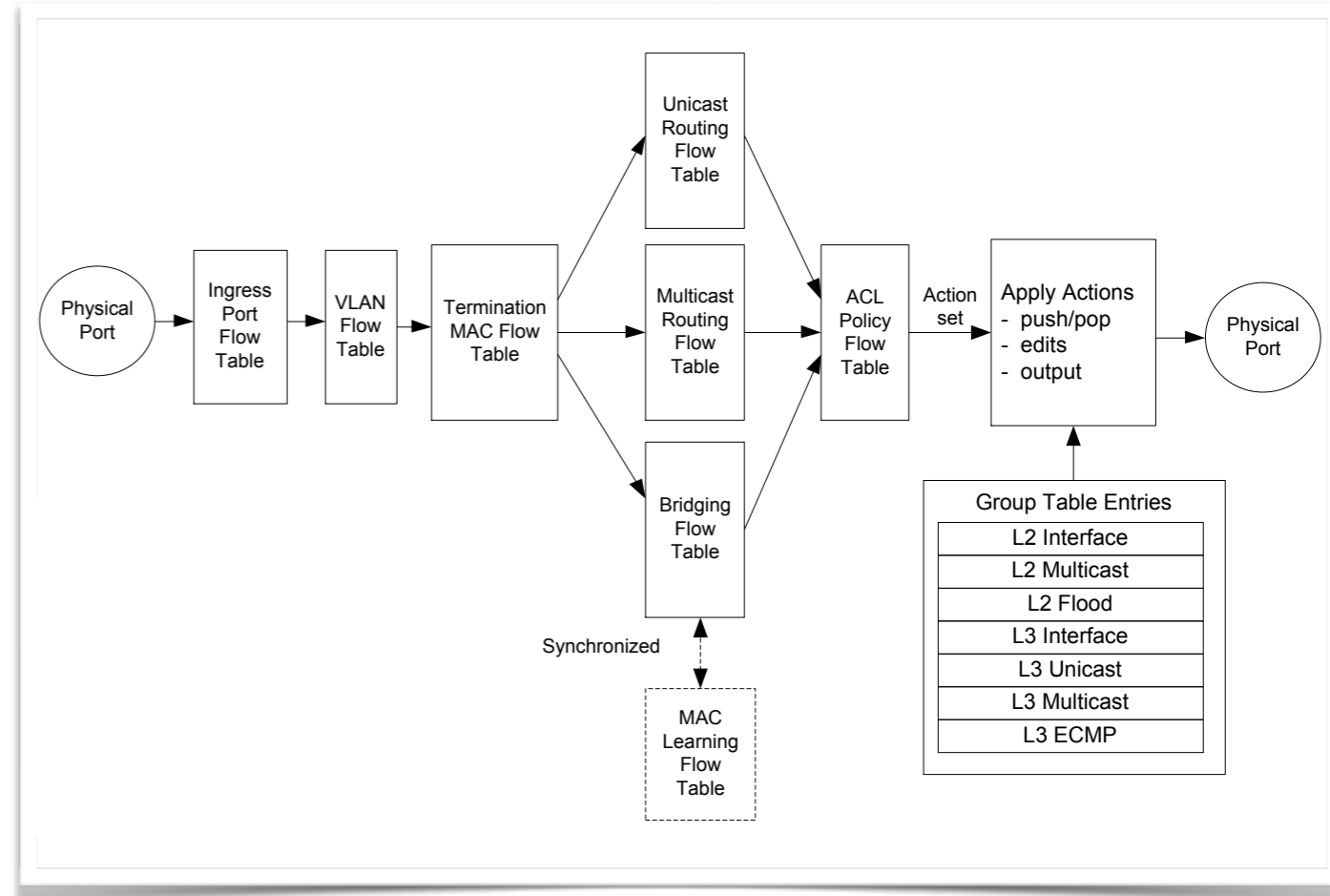
Data plane verification



- 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



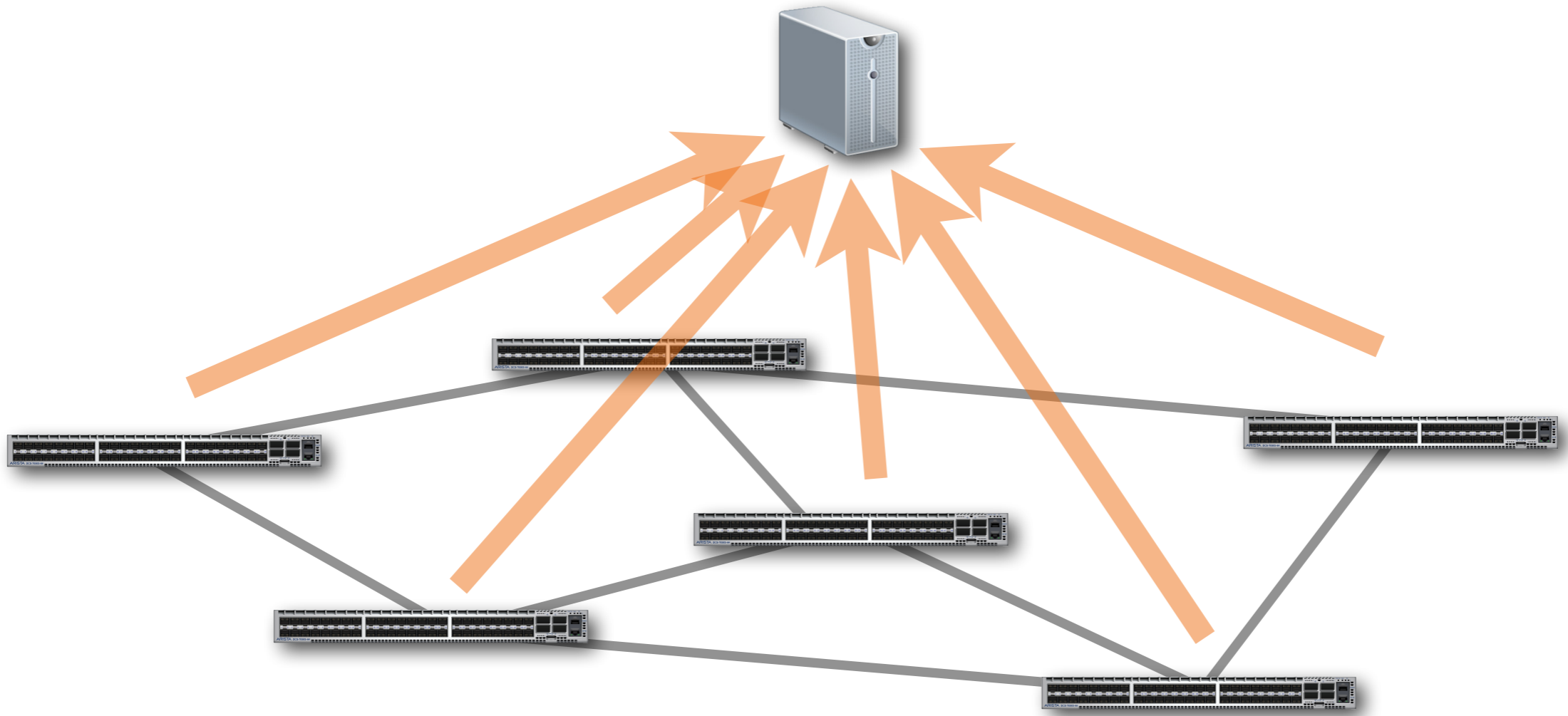
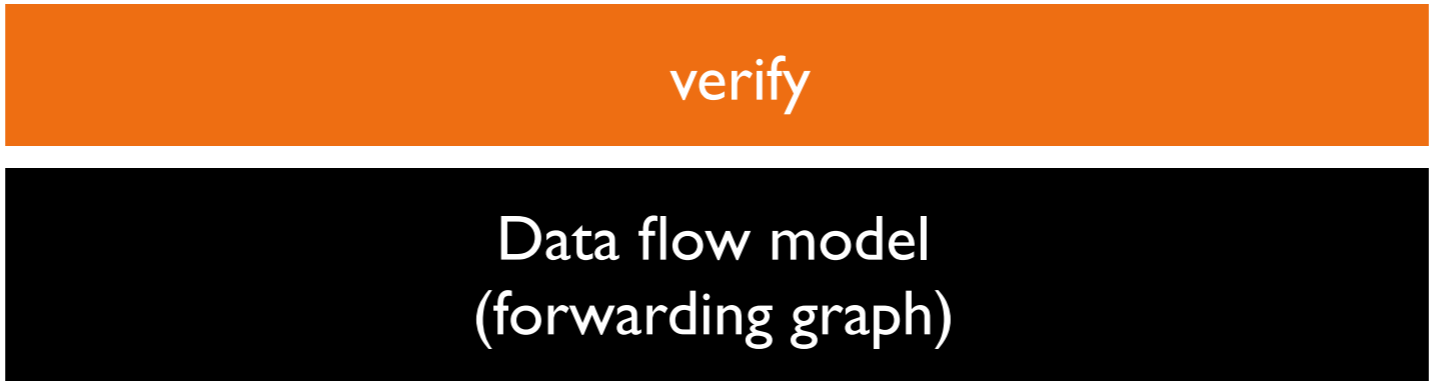
Broadcom's OF-DPA 1.0 Abstract Switch

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

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

- Some hope: Vendor-specific APIs, OpenConfig

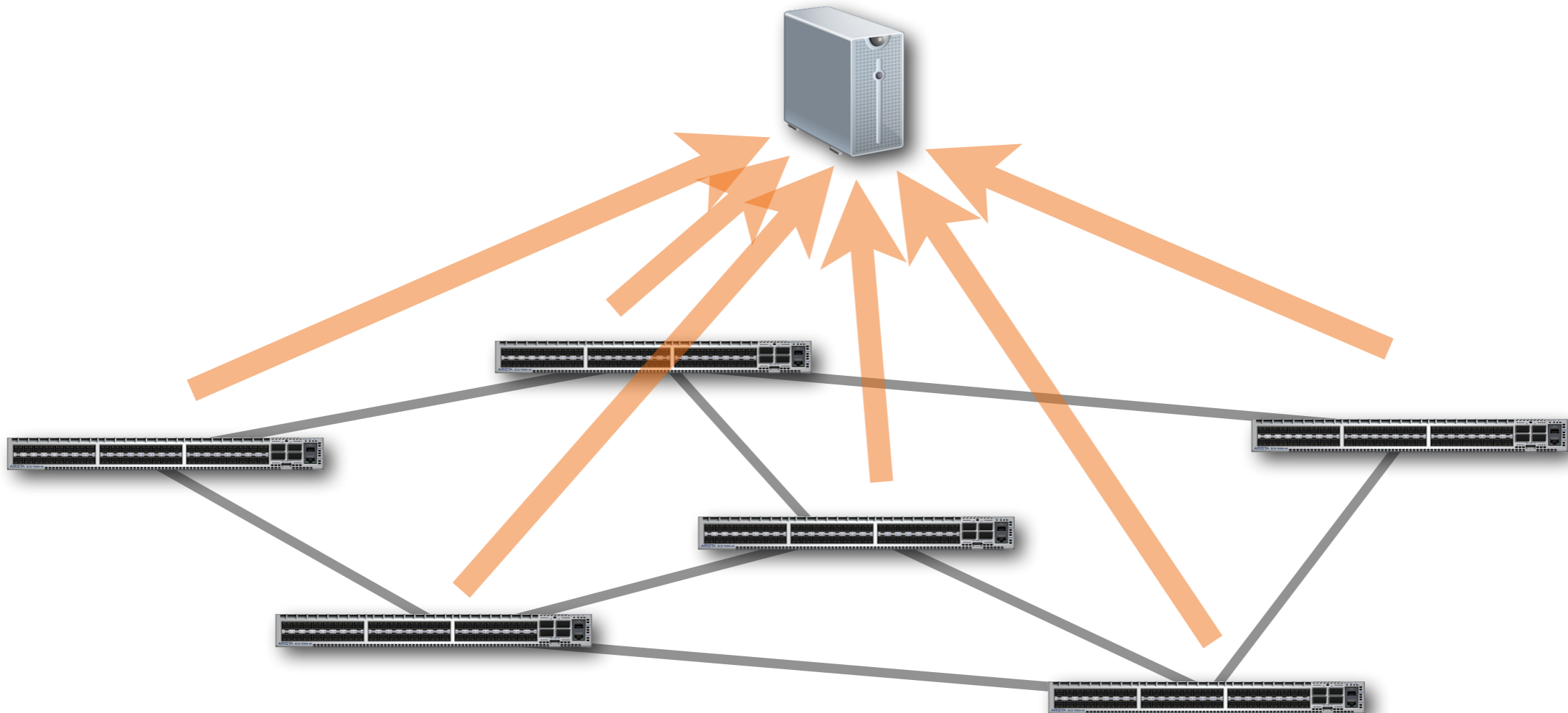
4. Model / Verifier separation works



4. Model / Verifier separation works

verify visualize search API optimize synthesize ...

Data flow model
(forwarding graph)



4. Model / Verifier separation works

Automatically fix correctness violations

NEAt: Network Error Auto-Correct
Wenxuan Zhou Jason Croft Bingzhe Liu Matthew Caesar
University of Illinois at Urbana-Champaign
{wzhou10, croft1, bingzhe, caesar}@illinois.edu

ABSTRACT
Configuring and maintaining an enterprise network is a challenging and error-prone process. Administrators must often consider security policies from a variety of sources simultaneously, including regulatory requirements, industry standards, and to mitigate attack vectors. Erroneous implementation of a policy, however, can result in costly data breaches and intrusions. Relying on humans to discover and troubleshoot such errors is not a sustainable solution. Modern enterprise networks must comply with highly stringent security demands, including regulatory requirements, or industry standards, such as PCI, HIPAA, and SOX. As a result, network administrators must carefully design and maintain their networks to follow those policies, by mapping out device contexts and access to sensitive resources, assessing risk, and installing access control policies that effectively mitigate that risk. However, mistakes and errors in implementing the policies can result

Automatically generate changes from spec

NetGen: Synthesizing Data-plane Configurations for Network Policies
Shambwaditya Saha Santhosh Prabhu P. Madhusudan
University of Illinois at Urbana-Champaign, USA.
{ssaha6,prabhum2,madhu}@illinois.edu

ABSTRACT
Networks are hard to maintain. When the physical network changes or when policies, most importantly security properties change, it is hard to change the network while maintaining all other existing policies. We study the problem of *network change synthesis*, where given a current network policies. The problem is exacerbated by the distributed nature of networks, which makes it difficult to make the correct changes, and when there are problems, to detect and debug them. Often, network operators make changes that appear to be right, only to discover later that they were not. Worse still, faulty configurations sometimes go unnoticed.

Optimize unoptimized SDN data planes (automatic fault tolerance etc)

Let me rephrase that: Transparent optimization in SDNs
Santhosh Prabhu* Mo Dong* Tong Meng P. Brighten Godfrey Matthew Caesar
University of Illinois, Urbana-Champaign
{prabhum2, modong2, tongm2, pbg, caesar}@illinois.edu

ABSTRACT
Enterprise networks today have highly diverse correctness requirements and relatively common performance objectives. As a result, preferred abstractions for enterprise networks are those which allow matching correctness specification, while transparently managing performance. Existing SDN network management architectures, however, bundle correctness and performance as a single abstraction. We argue that this creates an SDN ecosystem that is unnecessarily hard to build, maintain and evolve. We advocate a separation of concerns where correctness and performance are treated as separate concerns. The definition of correctness are extremely diverse. The definition of correctness, and the abstraction used to state it depends on many factors, such as the nature of the enterprise, traffic the network carries, applications that use the network, various compliance requirements, whether the network is a private data center or a campus or a WAN, etc. Furthermore, the active SDN research community is now experimenting with novel abstractions. Performance on the other hand is better understood, and there exist well-defined best practices to meet the

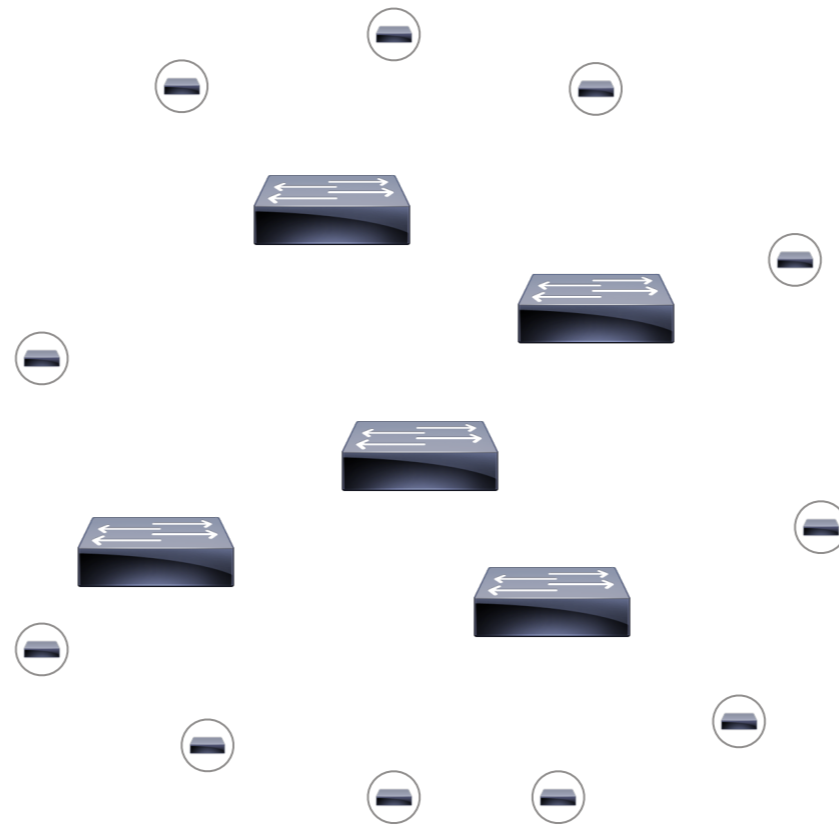
An Assertion Language for Debugging SDN Applications
Ryan Beckett, X. Kelvin Zou, Shuyuan Zhang, Sharad Malik, Jennifer Rexford, and David Walker
Princeton University
{rbeckett, xuanz, shuyuanz, sharad, jrx, dpw}@Princeton.edu

ABSTRACT
Software Defined Networking (SDN) provides opportunities for network verification and debugging by offering centralized visibility of the data plane. This has enabled both offline and online data-plane verification. However, little

1. INTRODUCTION
SDNs are a promising approach for managing network complexity. However, subtle bugs introduced from complicated software remain problematic. Network operators need sophisticated tools and techniques to proactively catch

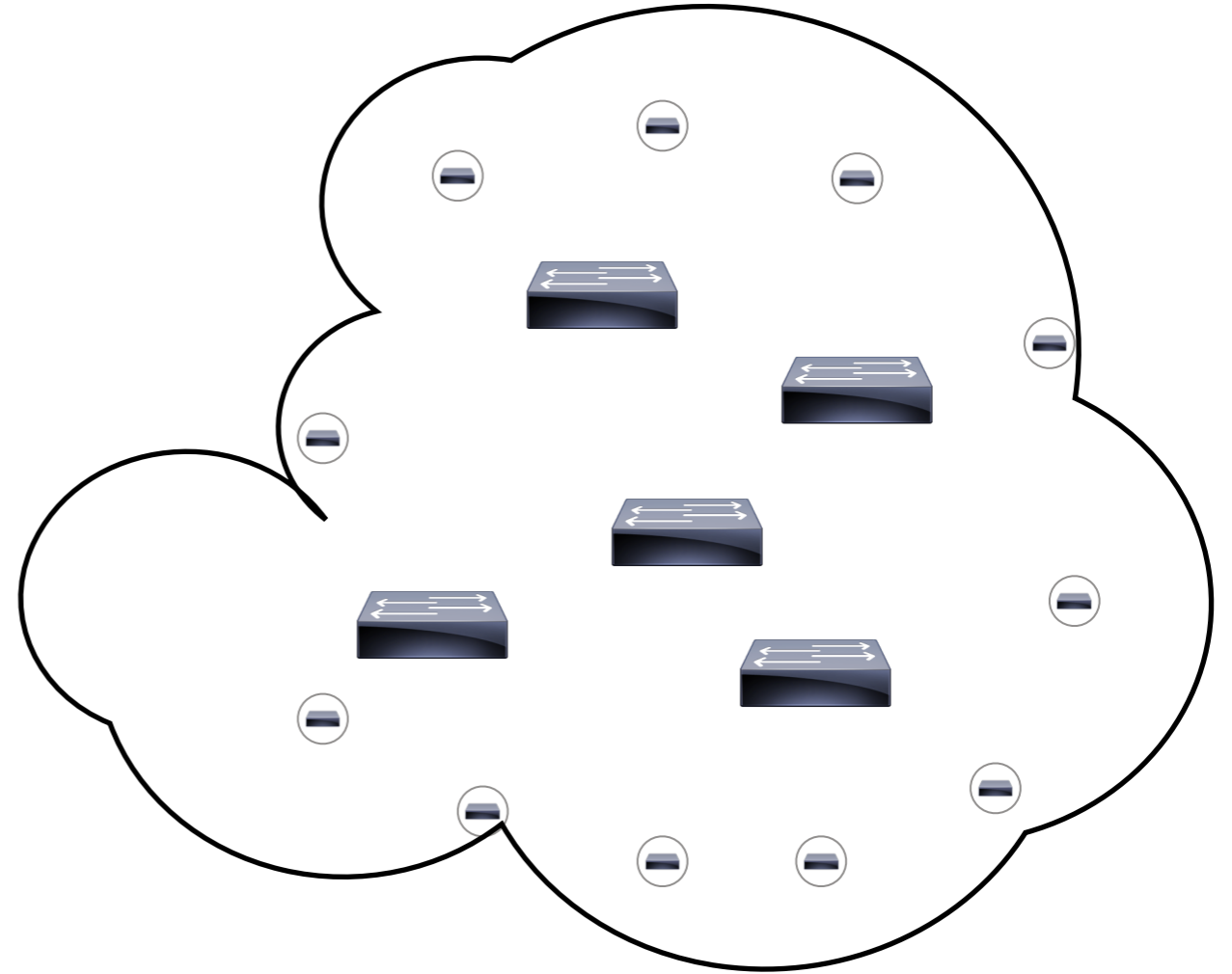
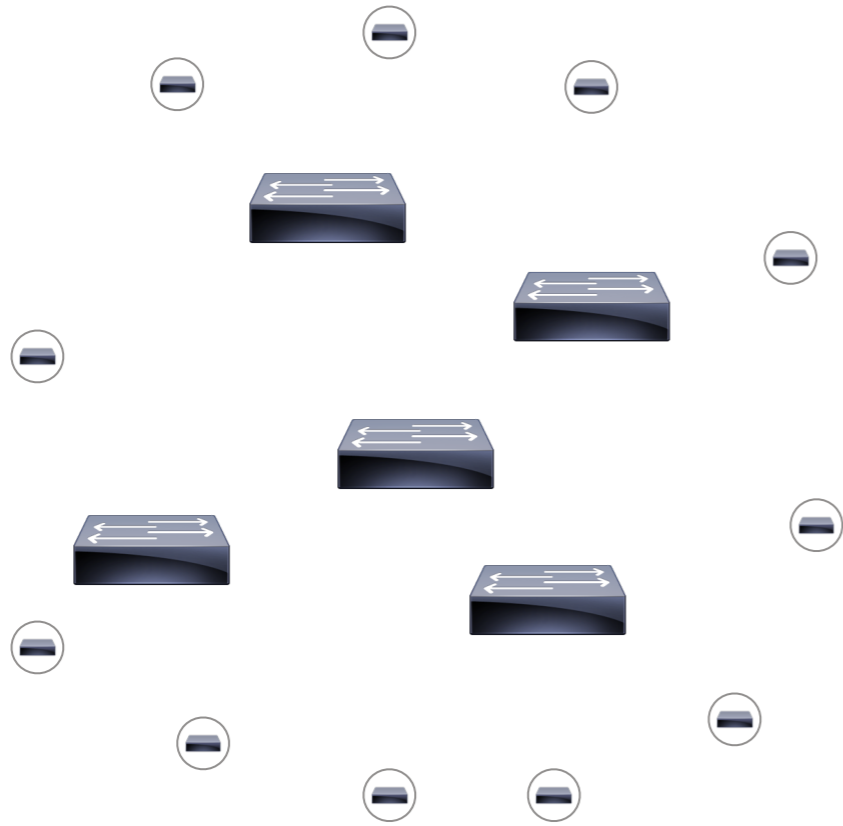
SDN Assertion Library

5. How do you know the intent?

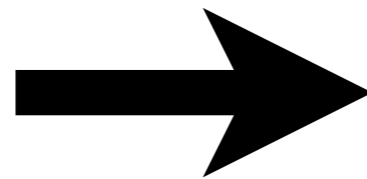


Traditional frame of mind:
Network as individual
devices with individual
config knobs

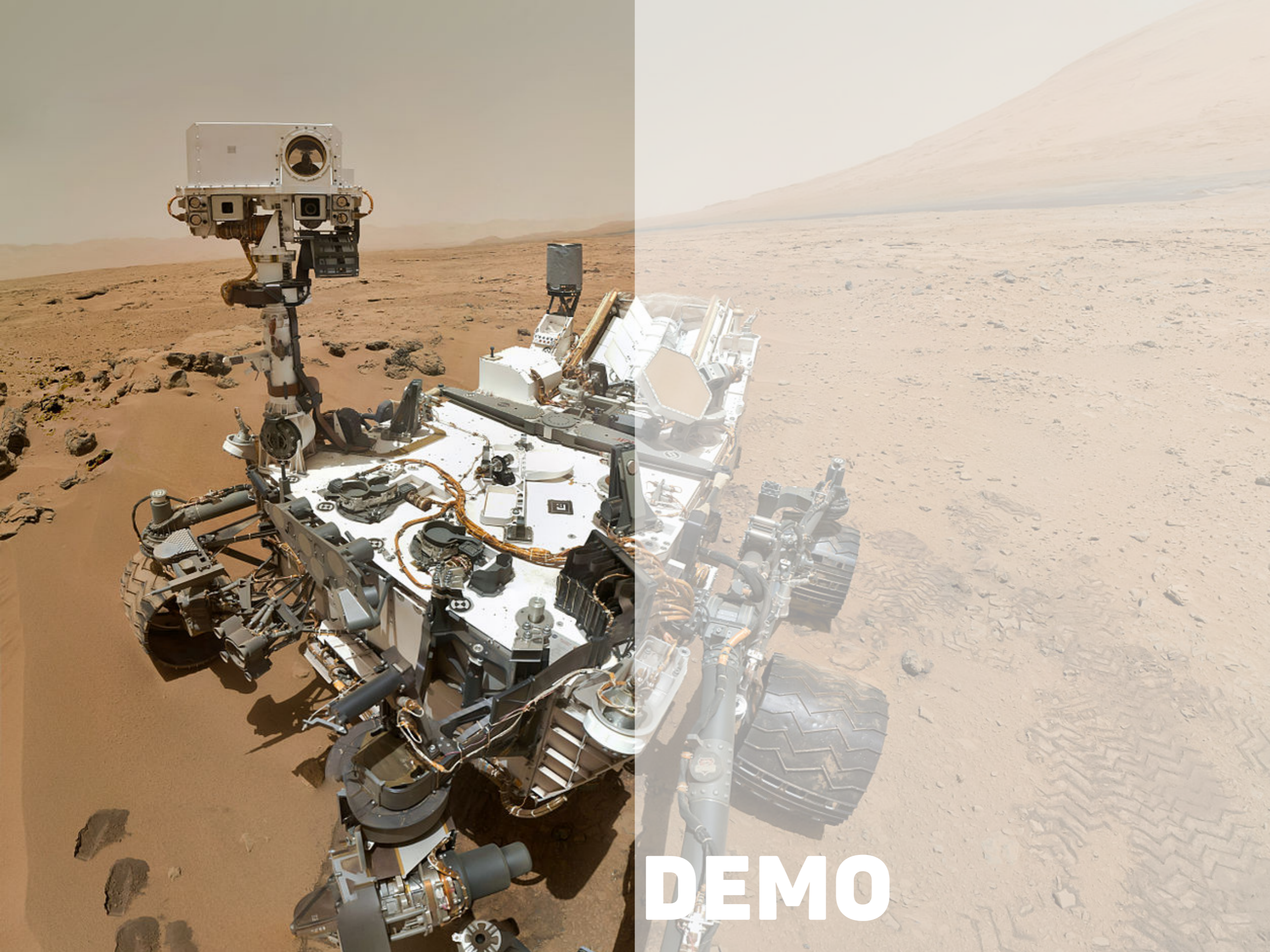
5. How do you know the intent?



Traditional frame of mind:
Network as individual
devices with individual
config knobs



Need: Network as
one system with
end-to-end intent



DEMO



THANK YOU!