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
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
Configs use many protocols & features

version 12.4
service timestamps debug datet ime msec
service timestamps log datet ime 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 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 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 GigabitEthernet0/2
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

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.X.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/](http://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]
Here, that the PC be valid. The PV replies with the PC \( \text{PV} \) where the form as RSAl The client then sends a request to the PV of and have each generated a public/private key pair e in Figure rl Before the process begins, the client and PV request sl and handshake with an appropriate PV for future rek. If the client uses a third parties, can exist in parallel. If the client uses a DNS servers are examples in today's Internet, not unprecedented certificate authorities and the root is that servers need to trust a third party. But this is delay for each new server or domain. The disadvantage the server directly. Thus, the first connection takes two level request to the server, thereby, it can contact the server to verify source provenance similar to the qWH's this section is to demonstrate a practical means for a client to verify the provenance certificate (PC) with a provenance verifier (PV) run by its domain at a known location. envision two common use cases that avoids replay attacks. nance in the request packet sent to servers, in a way multiple requests to place cryptographic proof of prove. taining this certificate once, the client can use it for cryptographic proof that the PV recently verified that the server is authentic. The PV may be any party trusted by the server. We 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 verification
- Controller verification & verifiable control languages
- Data plane verification
- Software switch verification
DATA PLANE VERIFICATION
Configuration verification

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”
“Can any packet starting at A reach B?”
A little calculation...

# theoretical packets

\[ = 2^{(#\text{header bits})} \times \#\text{injection points} \]
\[ = 2^{(18 \text{ byte ethernet} + 20 \text{ byte IPv4})} \times 10,000 \text{ ports} \]
\[ = 3.25 \times 10^{95} \text{ possible packets} \]

Estimated # atoms in observable universe

Grams of salt in the Dead Sea

\[ 3.7 \times 10^{15} \quad 10^{80} \quad 10^{95} \]
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: \( x[0] \ x[1] \ x[2] \ldots \ x[n] \)

\[
(x_4 \lor x_7 \lor \overline{x}_1) \land (\ldots) \land (\ldots) \land (\ldots)
\]

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 \( P(u, v) \)

<table>
<thead>
<tr>
<th>Destination</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.1.1.0/24</td>
<td>Fwd to ( V )</td>
</tr>
</tbody>
</table>

\[
P(u, v) = \text{dst\_ip} \in 10.1.1.0/24
\]
Reachability as SAT solving

Goal: reachability from $u$ to $w$

\[ C = (P(u, v) \land P(v, w)) \text{ is satisfiable} \]

- 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 $\Rightarrow$ boolean constraints over adjacent packet versions

$(p_i.dst_{-}ip \in 0.1.1.0/24) \land (p_{i+1}.label = 5)$

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

<table>
<thead>
<tr>
<th></th>
<th>Loop</th>
<th>Packet loss</th>
<th>Consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Being fixed</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Stale config.</td>
<td>0</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>Total alerts</td>
<td>9</td>
<td>17</td>
<td>2</td>
</tr>
</tbody>
</table>
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

Admin. interface

12.34.56.0/24
REAL-TIME DATA PLANE VERIFICATION
Not so simple

Challenge #1: Obtaining real time view

Challenge #2: Verify quickly
"Service $S$ reachable only through firewall?"
Here that the PC be valid. The PV replies with the PCx where the form as RSA. The client then sends a request to the PV of and have each generated a public/private key pair e in Figure rl Before the process begins, the client and PV handshake with an appropriate PV for future re-k. PV the server does not trust, it can fall back to a qWH third parties can exist in parallell. If the client uses a DNS servers are examples in today's Internet. Is that servers need to trust a third partyl But this is delay for each new server or domainl The disadvantage is that a client can avoid paying an RTT of web sites or content distribution networksl. That attract the same user frequently, such as popular a single RTTl This will be highly e. RTTs ease in TCP, and subsequent connections require the server directlyl Thus, the first connection takes two level request to the serverl and thereafter, it can contact a PC from the PV prior to initiating the applicationl. The first time a client contacts a domain, it obtains envision two common use casesl. In our implementation, however, each message would use TCP, thus proving provenance via TCP's guarantee, yet without introducing an RTT delayl. This section is to demonstrate a practical means for a server to verify source provenance similar to the qWH's.
Here that the PC be valid. The PV replies with the PCx where the form as RSA.

The client then sends a request to the PV of and have each generated a public/private key pair e in Figure r. Before the process begins, the client and PV handshake with an appropriate PV for future use.

DNS servers are examples in today's Internet. Not unprecedented, certificate authorities and the root is that servers need to trust a third party. But this is a delay for each new server or domain. The disadvantage is that a client can avoid paying an RTT for each web site or content distribution network. That attracts the same user frequently, such as popular web sites. A single RTT is highly efficient in TCP, and subsequent connections require the server directly. Thus, the first connection takes two RTTs easier in TCP.

Every architecture in the request packet sent to servers in a way containing this certificate once, the client can use it for client was reachable at a certain IP address. After obtaining a provenance certificate (RC), the PV run by its domain at a known location. E

 envision two common use cases. That avoids replay attacks. Provenance and Request Certificates and using them to establish a connection. Provenance and Request Certificates and using them to establish a connection.

4.2 Verifying provenance without a handshake

The protocol by which a client obtains a PC is shown. First, the PV may simply be the web server itself, or a different trusted party trusted by the server. We envision two common use cases that avoids replay attack. Lifecycle leverages cryptographic proof to verify the provenance of client requests without requiring an RTT when the PC becomes valid and

The obvious implementation of the above exchange is to demonstrate a practical means for a server to verify source provenance similar to the qWH's. In our implementation, however, each message would use TCP, thus proving provenance via TCP's cryptographic proof that the PV recently verified that the source address of the client, is the time the client sent the request in order to bypass the adversary to replay a connection request.

The PV replies with the PCx where the form as RSA. The client then sends a request to the PV of and have each generated a public/private key pair e in Figure r. Before the process begins, the client and PV handshake with an appropriate PV for future use.

DNS servers are examples in today's Internet. Not unprecedented, certificate authorities and the root is that servers need to trust a third party. But this is a delay for each new server or domain. The disadvantage is that a client can avoid paying an RTT for each web site or content distribution network. That attracts the same user frequently, such as popular web sites. A single RTT is highly efficient in TCP, and subsequent connections require the server directly. Thus, the first connection takes two RTTs easier in TCP.

Every architecture in the request packet sent to servers in a way containing this certificate once, the client can use it for client was reachable at a certain IP address. After obtaining a provenance certificate (RC), the PV run by its domain at a known location e.

envision two common use cases. That avoids replay attacks. Provenance and Request Certificates and using them to establish a connection. Provenance and Request Certificates and using them to establish a connection.

4.2 Verifying provenance without a handshake

The protocol by which a client obtains a PC is shown. First, the PV may simply be the web server itself, or a different trusted party trusted by the server. We envision two common use cases that avoids replay attack.

Provenance and Request Certificates and using them to establish a connection. Provenance and Request Certificates and using them to establish a connection.

4.2.2 Obtaining a Provenance Certificate

The protocol by which a client obtains a PC is shown. First, the PV may simply be the web server itself, or a different trusted party trusted by the server. We envision two common use cases that avoids replay attacks. Provenance and Request Certificates and using them to establish a connection. Provenance and Request Certificates and using them to establish a connection.

4.2.2 Obtaining a Provenance Certificate

The protocol by which a client obtains a PC is shown. First, the PV may simply be the web server itself, or a different trusted party trusted by the server. We envision two common use cases that avoids replay attacks. Provenance and Request Certificates and using them to establish a connection.
### Verifying invariants quickly

Find only equivalence classes affected by the update via a multidimensional trie data structure.
Verifying invariants quickly

Updates

Generate Equivalence Classes

Generate Forwarding Graphs

Veriflow

All the info to answer queries!
Verifying invariants quickly

Update:
- Generate Equivalence Classes
- Generate Forwarding Graphs
- Run Queries

Veriflow

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

Controller

Remove rule 1 (delayed)
Install rule 2

Rule 1
Rule 2

Switch A
Switch B

Possible network states:

One solution: “consistent updates”

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?

![Graph showing the number of rules in the network over time.](image-url)

- Immediate Update
- CCG
- Consistent Updates
- Completion Time

Legend:
- Immediate Update
- CCG
- Consistent Updates
- Completion Time
Challenges and Approach

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

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

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

//----------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
Stage 2: Compute data plane

**OspfExport**
- node=n2,
- network=10.0.0.0/24,
- cost=10,
- type=ospfE2).

**Fib**
- node=n1,
- network=10.0.0.0/24,
- egressInterface=int1_2).

**InstalledRoute**
- route=
  - node=n1,
  - network=10.0.0.0/24,
  - nextHop=n2
  - administrativeCost=110,
  - protocolCost=10,
  - protocol=ospfE2).
Stage 3: Data plane analysis

Counterexample of multipath consistency

```plaintext
{ 
  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 \rightarrow n2:int2_1 ],
  2nd hop:[ n2:int2_10 \rightarrow n10:int10_2 ],
  fate=accepted).

-----------------------------------------------

FlowPathHistory(
  flow={ node=n1, ..., dstIp=10.0.0.2 },
  1st hop:[ n1:int1_3 \rightarrow n3:int3_1 ],
  fate=nullRouted by n3).
New Consistency Properties

Multipath – disposition consistent on all paths

10.0.0.0/24

n1
n2
n3
n10
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
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

<table>
<thead>
<tr>
<th></th>
<th>Invariant</th>
<th>Total Violations</th>
<th>Violations Confirmed By Operators</th>
<th>Violations Fixed by Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Net1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multipath</td>
<td>32(4)</td>
<td>32(4)</td>
<td>21(3)</td>
</tr>
<tr>
<td></td>
<td>Failure</td>
<td>16(7)</td>
<td>3(2)</td>
<td>0(0)</td>
</tr>
<tr>
<td></td>
<td>Destination</td>
<td>55(6)</td>
<td>55(6)</td>
<td>1(1)</td>
</tr>
<tr>
<td><strong>Net2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multipath</td>
<td>11(3)</td>
<td>11(3)</td>
<td>11(3)</td>
</tr>
<tr>
<td></td>
<td>Failure</td>
<td>77(26)</td>
<td>18(7)</td>
<td>0(0)</td>
</tr>
</tbody>
</table>
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Data plane generation</th>
<th>Multipath consistency</th>
<th>Failure consistency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Net1</strong></td>
<td>238 min</td>
<td>75 x (&lt; 1.5 min)</td>
<td>199 x (~238 min)</td>
</tr>
<tr>
<td><strong>Net2</strong></td>
<td>37 min</td>
<td>17 x (&lt; 1.5 min)</td>
<td>279 x (~37 min)</td>
</tr>
</tbody>
</table>
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

- 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, 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]
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
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

Data plane verification

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

```
#include <stdio.h>

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

- 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

Some hope: Vendor-specific APIs, OpenConfig

Broadcom’s OF-DPA 1.0 Abstract Switch
Looking ahead: An Opportunity

Applications!

1. Network
   Routers, switches, firewalls, ...

2. Real time "knowledge layer"
   Formal model of network behavior

Snapshots or real-time stream of:
- Topology
- Data plane state (forwarding tables)

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!