The present and future of network verification

Ratul Mahajan
UW CSE 561, Winter 2021
“At least 41% of all calls that attempted to use T-Mobile’s network during the outage failed, including at least 23,621 failed calls to 911.”

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

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

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

Guarantee network behavior

* Some aspect of behavior
† Under some assumptions
A horizontal slice of the problem

Verify

Trust

Configuration, state

Software (OS, protocols)

Hardware
The “haystack” of network behaviors is HUGE

**Large scale**
- \(O(10^3)\) devices
- \(O(10^4)\) config lines / device
- \(O(10^6)\) FIB entries / device

**Complex interactions**
- Distributed routing
- Protocol redistribution
- Rich route filters
The “needles” in network behavior are tiny

Only specific route announcements
Only specific failures
Only specific packets
Only specific announcement ordering
The 2D space of network verification tools

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

Features:
- Data plane verification
- Control plane verification

# states analyzed:
- One (live) state
- Some states
- All states
Data plane verification

Who can talk to whom using which packets and paths in one state of the network?
Can A talk to D and using which packets?
DPV idea: Ternary simulation

Union packet sets along possible paths
Solve using custom data structure or BDDs

HSA [2012]
More DPV

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

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

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

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

Programmable data planes: p4v [2018]

Stateless DPV is a “solved problem”
Stateful and programmable DPV not there yet
Control plane verification

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

- Finds bugs proactively
- Enables what if analysis
Verifying distributed control planes

Routers generate and process messages per low-level directives

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

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

Goal

Reason about states that emerge when many such programs run concurrently
CPV idea #1: Simulate the control plane

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

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

Batfish [2015]
CPV idea #2: Encode the fixed point

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

ARC [2016] use a graph encoding (not general)
Minesweeper [2017] uses SMT encoding
Minesweeper overview

“Does P hold in the network?”

Network encoding: \( N \)

\[ \land \]

Property: \( \neg P \)

Satisfiable: Property violation

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

Circuit view for R1_{BGP}

Redistribution From CON to BGP

BGP peering Between R1 and R2

After R2 export filter

After R1 import filter
Encode routing messages

Routing Record

- valid: 1 bit
- prefix: \([0, 2^{32})\)
- prefixLen: \([0, 2^5)\)
- adminDist: \([0, 2^8)\)
- localPref: \([0, 2^{32})\)
- metric: \([0, 2^{32})\)
- med: \([0, 2^{32})\)
- ospfType: \([0, 2^2)\)
2 Encode routing messages

Import filter on R1 from R2
- ip prefix-list PL1 deny 192.168.0.0/16 le 32
- ip prefix-list PL1 allow
- route-map FromR2 10
  - match ip address prefix-list PL1
  - set local-preference 120

If R2 exports a route and it passes the import filter:
- Then R1 has the same route with local preference of 120

Otherwise, R1 has no route from R2

If \( e_4 \) valid and failed\( R_{1,R2} = 0 \) and
\( \neg (\text{FBM}(e_4 \text{.prefix}, 192.168.0.0, 16) \land 16 \leq e_4 \text{.prefixLen} \leq 32) \) then
- \( \text{in}_4 \text{.valid} = \text{true} \)
- \( \text{in}_4 \text{.lp} = 120 \)
- \( \text{in}_4 \text{.prefix} = e_4 \text{.prefix} \)
- \( \text{in}_4 \text{.metric} = e_4 \text{.metric} \)
- \( \text{in}_4 \text{.prefixLen} = e_4 \text{.prefixLen} \)

... else \( \text{in}_4 \text{.valid} = \text{false} \)
What is the best BGP route?

\[ R_{1\text{BGP}}\text{-BEST} = \text{Min}(\text{in1, in4, in5, in7}) \]

What is the best overall route?

\[ R_{1\text{-BEST}} = \text{Min}(R_{1\text{BGP}\text{-BEST}}, R_{1\text{OSPF}\text{-BEST}}, R_{1\text{CON}\text{-BEST}}) \]
Encode routing decisions

Does R1 have a RIB entry for R2?
$$RIB_{R1,R2} = (\text{in}4 = \text{R1-Best})$$

Does R1 have a FIB entry for R2?
$$FIB_{R1,R2} = RIB_{R1,R2} \land \neg ACL(R1,R2)$$
Encode the data packet

Symbolic Packet

\[
\begin{align*}
0 & \leq \text{dstIp} \leq 2^{32} \\
0 & \leq \text{srcIp} \leq 2^{32} \\
0 & \leq \text{dstPort} \leq 2^{16} \\
0 & \leq \text{srcPort} \leq 2^{16} \\
0 & \leq \text{protocol} \leq 2^{8}
\end{align*}
\]
Encode the property

Can R3 reach subnet S2?

\[
\text{canReach}_{R_2} \iff \text{FIB}_{R_2, S_2}
\]

\[
\text{canReach}_{R_1} \iff (\text{FIB}_{R_1, R_2} \land \text{canReach}_{R_2}) \lor (\text{FIB}_{R_1, R_3} \land \text{canReach}_{R_3})
\]

\[
\text{canReach}_{R_3} \iff (\text{FIB}_{R_3, R_1} \land \text{canReach}_{R_1})
\]

Property: \( \text{canReach}_{R_3} \)
Optimizing the encoding

Must use domain knowledge

SMT solvers cannot automagically perform even “straightforward” optimizations
Prefix Hoisting

**Routing Record**

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

FBM\( (r.\text{prefix}, 192.4.0.0, 16) \cap 16 \leq r.\text{prefixLen} \leq 32 \)

\[ \implies \]

FBM\( (\text{dstlp}, 192.4.0.0, 16) \cap 16 \leq r.\text{prefixLen} \leq 32 \)

\[ \implies \]

\((192.4.0.0 \leq \text{dstlp} \leq 192.4.0.0 + 2^{32-16}) \cap 16 \leq r.\text{prefixLen} \leq 32 \)

Integer Difference Logic!
Slicing

### Symbolic Record

- **valid**: 1 bit
- **prefixLen**: \([0,2^5)\)
- **adminDist**: \([0,2^8)\)
- **localPref**: \([0,2^{32})\)
- **metric**: \([0,2^{32})\)
- **med**: \([0,2^{32})\)
- **ospfType**: \([0,2^2)\)

#### Statically analyze configs to infer irrelevant attributes

Local preference has no impact if never explicitly set.
Does it work for real networks?

152 networks
OSPF, eBGP, iBGP
Static routes
ACLs
Route redistribution

120 violations
Hijack-able internal addresses
Asymmetric paired ACLs

Proved fault tolerance
How well does it scale?

< 5 minutes
More CPV

Tradeoff generality for scalability
  • ARC [2016], Tiramisu [2019], FastPlane [2019], Shapeshifter [2020]

Model checking
  • Plankton [2019]

Exploit symmetry
  • Bonsai [2018], Origami [2019]

Fully general CPV that scales to thousands of nodes is still an open problem
Reflections on research to practice journey
Remarkably short path from research to practice

All large cloud providers are using it

Startups are enabling broader use
Research to practice gap (1/3)

Network verification is “too complete”
Flags uninteresting violations

Real networks have unwritten assumptions that cause violations

- Devices cannot spoof source IPs
- Blocking “malicious” IPs or telnet ports is OK
- Loosing connectivity under specific failures is OK
Research to practice gap (2/3)

Network verification answers are “too precise”
Hard to explain results to users

Oddly-shaped packet header spaces and network environments
Not easy to map back to user-configured objects
Network verification is “too different”
Hard for network engineers to use

Testing with concrete inputs is easier than specifying invariants

Inversion of mental model is needed
The future of network verification
Network verification 1.0

Identify promising slices of the problem

Develop core techniques

Network verification 2.0

Enable effective use in practice

Enable rapid expansion to more functionality
Verified networks can have outages too.

- Configuration change
- check invariants
- untested network
Verified networks can have outages too

Route maintenance withdraw /18

BGP routes look correct

Traffic carried by untested static route and dropped by firewall
Inspiration from software: Code coverage

<table>
<thead>
<tr>
<th>Files</th>
<th>Complexity</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>allinone/src/main/java/org/batfish/allinone</td>
<td>52.38%</td>
<td>62.91%</td>
</tr>
<tr>
<td>batfish-client/src/main/java/org/batfish/client</td>
<td>61.10%</td>
<td>64.57%</td>
</tr>
<tr>
<td>batfish-common-protocol/src/main/java/org/batfish</td>
<td>70.02%</td>
<td>78.02%</td>
</tr>
<tr>
<td>batfish/src/main/java/org/batfish</td>
<td>62.26%</td>
<td>70.63%</td>
</tr>
<tr>
<td>coordinator/src/main/java/org/batfish/coordinator</td>
<td>63.00%</td>
<td>65.27%</td>
</tr>
<tr>
<td>minesweeper/src/main/java/org/batfish/minesweeper</td>
<td>61.71%</td>
<td>72.96%</td>
</tr>
<tr>
<td>question/src/main/java/org/batfish/question</td>
<td>71.06%</td>
<td>81.47%</td>
</tr>
</tbody>
</table>
How to define “network coverage”?

Unlike programs, network is not a sequence of statements.

Model the network as a dependency graph of “facts”.

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

Beckett and Mahajan [2019]
Prototype deployed at Microsoft

- MethodName
- CoverageSummary
  - DeviceLevelCoverage: 100.00% (14/14)
  - FibRuleLevelCoverage: 39.39% (52/132)
  - FibRuleLevelMatchFieldCoverage: 39.39% (52/132)
  - FibRuleLevelActionFieldCoverage: 39.39% (52/132)
  - InterfaceLevelCoverage: 80.00% (64/80)
  - PathCoverage: 73.58% (39/53)
The future of network verification

Effective use in practice

Rapid expansion to more functionality
Network stack is broad and deep
Current monolithic approach does not scale

**Header Space Analysis**
Stateless forwarding

**Batfish**
Distributed routing

**Minesweeper**
Distributed routing
Inspiration from software: Intermediate languages

Front-end

Intermediate Language

Back-end

VCC  Dafny  Chalice  ...

Boogie

Z3  CVC  Yices
Zen: An IL for network verification

Stateless forwarding
Encapsulation
Middleboxes
Distributed routing
Virtual Networks

Ternary simulation
Concrete simulation
Stable path constraints
Model checking

Datalog
BDDs, custom
SMT solver
...

Zen [2020]
https://github.com/microsoft/zen
Modeling an Azure security group in Zen

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

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

Network verification is needed to guarantee correctness for increasingly complex networks.

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

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