A Hypothesis Testing Framework for Network Security

P. Brighten Godfrey
University of Illinois at Urbana-Champaign

TSS Seminar, September 15, 2015
Part of the SoS Lablet with

David Nicol

Kevin Jin  Matthew Caesar  Bill Sanders
Work with...

Anduo Wang
Wenxuan Zhou
Dong Jin
Jason Croft
Matthew Caesar

Ahmed Khurshid
Haohui Mai
Xuan Zhou
Rachit Agarwal
Sam King


Background: Network Verification
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]
Understanding your network

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

Configuration verification

e.g.: RCC for BGP [Feamster & Balakrishnan, NSDI’05]
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

- (Checks current snapshot)
- Insensitive to control protocols
- Accurate model
“Service S reachable only through firewall?”

“Is segment isolated?”
Building It
Verification is nontrivial

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

$A$

$x[4] = 1$

$x[7] = 0$

$B$

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

NP-complete!
Anteater’s solution

Express data plane and invariants as SAT

• ...up to some max # hops

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

$P(u, v) = \text{dst}_\text{ip} \in 10.1.1.0/24$
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 ⇒ boolean constraints over adjacent packet versions

\[(p_i \cdot dst\_ip \in 0.1.1.0/24) \land (p_{i+1} \cdot label = 5)\]

More generally: \(p_{i+1} = f(p_i)\)
Experience with an operational network
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 bugs with 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
Bugs found by other invariants

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
Can we verify networks in real time?
Challenge #1: Obtaining real time view of network

Challenge #2: Verification speed
“Service S reachable only through firewall?”

Verifier

Diagnosis
4.2 Verifying provenance without a handshake

4.2.2 Obtaining a Provenance Certificate

The protocol by which a client obtains a PC is shown in Figure 4. To do this, the client begins by constructing a request to the PV of a domain to which it has already connected. The PV replies with the PC, which is a set of information that includes the client's public key, the current time, and the duration for which the client requests the PC. The PC corresponds to cryptographic proof that the PV recently verified that the client was reachable at a certain IP address. After obtaining the PC, the client can use it to verify the provenance of client requests without requiring an RTT delay. The advantage of this design over TCP is that the PV can set the PC to remain valid for a certain period of time, which makes it more difficult for adversaries to replay connection requests. The PV can thus verify source provenance similar to the qWH's and include the protocol number, source and destination IP address, source port number, destination port number, and additional metadata such as the application number, data, and duration. The advantage is that a client can avoid paying an RTT delay on every connection. First, the client handshakes with an appropriate PV for future requests. If the client uses a DNS server for validation, the server does not trust the PV, it can fall back to a third-party certificate authority and the root server, which is that servers need to trust a third party. But this is less vulnerable to DoS attacks and more efficient than trusting a single point of failure. The disadvantage is that the PV may simply be the web server itself, or a trusted third party that can run a PV service and multiple domains. The PV may be any party trusted by the server. We discuss in this section is to demonstrate a practical means for a third party to obtain a PC from the PV prior to initiating the application. The obvious implementation of the above exchange would use TCP, thus proving provenance via TCP's abstractions. In our implementation, however, each message is a secure hash function that avoids replay attacks, and the client can receive messages at the same user frequently, such as popular websites or games. The obvious implementation of the above exchange would use TCP, thus proving provenance via TCP's abstractions. In our implementation, however, each message is a secure hash function that avoids replay attacks, and the client can receive messages at the same user frequently, such as popular websites or games. The obvious implementation of the above exchange would use TCP, thus proving provenance via TCP's abstractions. In our implementation, however, each message is a secure hash function that avoids replay attacks, and the client can receive messages at the same user frequently, such as popular websites or games.
VeriFlow architecture

Logically centralized controller

Thin, standard interface to data plane (e.g. OpenFlow)
Verifying invariants quickly

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

Veriflow

Generate Equivalence Classes

Generate Forwarding Graphs

Updates

All the info to answer queries!
Verifying invariants quickly

Veriflow

Generate Equivalence Classes

Generate Forwarding Graphs

Run Queries

Diagnosis report

- Type of invariant violation
- Affected set of packets
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
Towards a Science of Security: Network Hypothesis Testing
1. Modeling dynamic networks
2. Networks as databases
3. Provably correct virtual networks
Modeling dynamic networks
Timing uncertainty

One solution: “consistent updates”  
Uncertainty-aware verification

“certain”

“uncertain”
Update synthesis via verification

Enforcing dynamic correctness with heuristically maximized parallelism
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?
Software-defined Networks as Databases
Software-Defined Networks

Logically centralized controller

Thin, standard interface to data plane (e.g. OpenFlow)

Software abstractions

app | app

Logical centralized controller

Thin, standard interface to data plane (e.g. OpenFlow)

Software abstractions

app | app

Logical centralized controller

Thin, standard interface to data plane (e.g. OpenFlow)
Ravel: database view of net control

- Network
- Base tables
- Event notification
- Query, update
- App view
- Standard SQL database
- Openflow control
- Network
Ravel example

balance load → verify

load balancer → access control

tenant virtual network → shortest path

compute path

3
tenant virtual network

view

view

base

add_flow  del_flow

traffic matrix → configuration

compute path
Key benefits

Abstraction via SQL

Orchestration via data-sharing

“Bonus” DB services

- verification, synthesis via view maintenance, update
- transaction processing
Impact of Network Verification
Configuration verification


Firewall verification

- **Margrave** [Nelson, Barratt, Dougherty, Fisler, Krishnamurthi, LISA’10]
Data plane verification

- **Static reachability in IP networks** [Xie’05]
- **FlowChecker** [Al-Shaer, Al-Haj, SafeConfig ’10]
- **ConfigChecker** [Al-Shaer, Al-Saleh, SafeConfig ’11]

- **Anteater** [Mai, Khurshid, Agarwal, Caesar, G., King, SIGCOMM’11]
- **VeriFlow** [Khurshid, Zou, Zhou, Caesar, G., HotSDN’12, NSDI’13]
- **CCG** [Zhou, Jin, Croft, Caesar, G., NSDI’15]

- **Header Space Analysis** [Kazemian, Varghese, and McKeown, NSDI ’12]
- **NetPlumber** [Kazemian, Chang, Zeng, Varghese, McKeown, Whyte, NSDI ’13]
- **Batfish** [Fogel, Fung, Pedrosa, Walraed-Sullivan, Govindan, Mahajan, Millstein, NSDI’15]
Checking Beliefs in Dynamic Networks
Nuno P. Lopes, Nikolaj Bjørner, and Patrice Godefroid, Microsoft Research; Karthick Jayaraman, Microsoft Azure; George Varghese, Microsoft Research

https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/lopes
Future research: Richer models

Software pipelines

Stateful Networks

Verifiable SDN Controllers

Higher layer concepts
(roles, people, applications)
Thanks!