Automatic Test Packet Generation

James Hongyi Zeng
with Peyman Kazemian,
George Varghese, Nick McKeown
Stanford University, UCSD, Microsoft Research

http://eastzone.github.com/atpg/

CoNEXT 2012, Nice, France
CS@Stanford Network Outage

Tue, Oct 2, 2012 at 7:54 PM:

“Between 18:20-19:00 tonight we experienced a complete network outage in the building when a loop was accidentally created by CSD-CF staff. We're investigating the exact circumstances to understand why this caused a problem, since automatic protections are supposed to be in place to prevent loops from disabling the network.”
Outages in the Wild

Hosting.com's New Jersey data center was taken down on June 1, 2010, igniting a cloud outage and connectivity loss for nearly two hours... Hosting.com said the connectivity loss was due to a software bug in a Cisco switch that caused the switch to fail.

On April 26, 2010, NetSuite suffered a service outage that rendered its cloud-based applications inaccessible to customers worldwide for 30 minutes... NetSuite blamed a network issue for the downtime.

The Planet was rocked by a pair of network outages that knocked it offline for about 90 minutes on May 2, 2010. The outages caused disruptions for another 90 minutes the following morning.... Investigation found that the outage was caused by a fault in a router in one of the company's data centers.
Network troubleshooting a problem?

• Survey of NANOG mailing list (June 2012)
  – Data set: 61 responders: 23 medium size networks (<10K hosts), 12 large networks (< 100K hosts)
  – Frequency: 35% generate >100 tickets per month
  – Downtime: 25% take over an hour to resolve. (estimated $60K-110K/hour [1])
  – Current tools: Ping, Traceroute, SNMP
  – 70% asked for better tools, automatic tests

The Battle

Hardware
Buffers, fiber cuts, broken interfaces, mis-labeled cables, flaky links

Software
firmware bugs, crashed module

VS

ping, traceroute, SNMP, tcpdump

+ wisdom and intuition
Automatic Test Packet Generation

Goal: automatically generate test packets to test the network state, and pinpoint faults before being noticed by application.

Augment human wisdom and intuition.
Reduce the downtime.
Save money.

Non-Goal: ATPG cannot explain why forwarding state is in error.
ATPG Workflow

FIBs, ACLs
Topology

ATPG

Test Packets

Network

Test Results
Systematic Testing

• Comparison: chip design
  – Testing is a billion dollar market
  – ATPG = Automatic Test Pattern Generation
Roadmap

- Reachability Analysis
- Test packet generation and selection
- Fault localization
- Implementation and Evaluation
Reachability Analysis

- Header Space Analysis (NSDI 2012)

- All-pairs reachability: Compute all classes of packets that can flow between every pair of ports.
Example

Box A

\[ r_{A1}, r_{A2}, r_{A3} \]

Box C

\[ r_{C1}, r_{C2} \]

Box B

\[ r_{B1}, r_{B2}, r_{B3}, r_{B4} \]
All-pairs reachability

<table>
<thead>
<tr>
<th>Header</th>
<th>Ingress Port</th>
<th>Egress Port</th>
<th>Rule History</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 )</td>
<td>( P_A )</td>
<td>( P_B )</td>
<td>( r_{A1}, r_{B3}, r_{B4}, \text{ link AB} )</td>
</tr>
<tr>
<td>( p_2 )</td>
<td>( P_A )</td>
<td>( P_C )</td>
<td>( r_{A2}, r_{C2}, \text{ link AC} )</td>
</tr>
<tr>
<td>( p_3 )</td>
<td>( P_B )</td>
<td>( P_A )</td>
<td>( r_{B2}, r_{A3}, \text{ link AB} )</td>
</tr>
<tr>
<td>( p_4 )</td>
<td>( P_B )</td>
<td>( P_C )</td>
<td>( r_{B2}, r_{C2}, \text{ link BC} )</td>
</tr>
<tr>
<td>( p_5 )</td>
<td>( P_C )</td>
<td>( P_A )</td>
<td>( r_{C1}, r_{A3}, \text{ link BC} )</td>
</tr>
<tr>
<td>( p_6 )</td>
<td>( P_C )</td>
<td>( P_B )</td>
<td>( r_{C1}, r_{B3}, r_{B4}, \text{ link BC} )</td>
</tr>
</tbody>
</table>
New Viewpoint: Testing and coverage

• HSA represents networks as chips/programs
• Standard testing finds inputs that cover every gate/flipflop (HW) or branch/function (SW)
New Viewpoint: Testing and coverage

- In networks, packets are inputs, different covers
  - Links: packets that traverse every link
  - Queues: packets that traverse every queue
  - Rules: packets that test each router rule

- Mission impossible?
  - testing all rules **10 times per second** needs **< 1% of link overhead** (Stanford/Internet2)
Roadmap

• Reachability Analysis
• Test packet generation and selection
• Fault localization
• Implementation and Evaluation
All-pairs reachability and covers

<table>
<thead>
<tr>
<th>Header</th>
<th>Ingress Port</th>
<th>Egress Port</th>
<th>Rule History</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 ) ( \text{dst_ip}=10.0/16, \ tcp=80 )</td>
<td>( P_A )</td>
<td>( P_B )</td>
<td>( r_{A1}, r_{B3}, r_{B4} ), link AB</td>
</tr>
<tr>
<td>( p_2 ) ( \text{dst_ip}=10.1/16 )</td>
<td>( P_A )</td>
<td>( P_C )</td>
<td>( r_{A2}, r_{C2} ), link AC</td>
</tr>
<tr>
<td>( p_3 ) ( \text{dst_ip}=10.2/16 )</td>
<td>( P_B )</td>
<td>( P_A )</td>
<td>( r_{B2}, r_{A3} ), link AB</td>
</tr>
<tr>
<td>( p_4 ) ( \text{dst_ip}=10.1/16 )</td>
<td>( P_B )</td>
<td>( P_C )</td>
<td>( r_{B2}, r_{C2} ), link BC</td>
</tr>
<tr>
<td>( p_5 ) ( \text{dst_ip}=10.2/16 )</td>
<td>( P_C )</td>
<td>( P_A )</td>
<td>( r_{C1}, r_{A3} ), link BC</td>
</tr>
</tbody>
</table>
Test Packet Selection

• Packets in all-pairs reachability table are more than necessary

• Goal: select a minimum subset of packets whose histories cover the whole rule set

A Min-Set-Cover problem
# Min-Set-Cover

<table>
<thead>
<tr>
<th>Packets</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>D</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>G</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packets</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>G</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Test Packets Selection

• Min-Set-Cover
  – Optimization is NP-Hard
  – Polynomial approximation (O(N^2))

- Exercise all rules
- Sent out periodically

- “Redundant”
- Will be used in fault localization
Roadmap

• Reachability analysis
• Test packet generation and selection
• Fault localization
• Evaluation: offline (Stanford/Internet2), emulated network, experimental deployment
Fault Localization

Stanford Backbone
Fault Localization

- Network Tomography? → Minimum Hitting Set
- In ATPG: we can choose packets!
- Step 1: Use results from regular test packets
  - $F$ (potentially broken rules) = Union from all failing packets
  - $P$ (known good rules) = Union from all passing packets
  - Suspect Set = $F - P$
Fault Localization

• Step 2: Use reserved test packets
  – Pick packets that test only one rule in the suspect set, and send them out for testing
  – Passed: eliminate
  – Failed: label it as “broken”

• Step 3: (Brute force…) Continue with test packets that test two or more rules in the suspect set, until the set is small enough
Roadmap

• Reachability analysis
• Test packet generation and selection
• Fault localization
• Implementation and Evaluation
Putting them all together

All-pairs Reachability Table

<table>
<thead>
<tr>
<th>Header</th>
<th>In Port</th>
<th>Out Port</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>10xx...</td>
<td>1</td>
<td>2</td>
<td>$R_1, R_5, R_{20}$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Header Space Analysis

Topology, FIBs, ACLs, etc

1. Test Terminal
2. Parser
3. Transfer Function
4. All-pairs Reachability
5. Fault Localization
Implementation

• Cisco/Juniper Parsers
  – Translate router configuration files and forwarding tables (FIB) into Header space representation

• Test Packet Generation/Selection
  – Hassel: A python header space library
  – Min-Set-Cover
  – Python’s `multiprocess` module to parallelize

• SDN can simplify the design
Datasets

- Stanford and Internet2
  - Public datasets
- Stanford University backbone
  - ~10,000 HW forwarding entries (compressed from 757,000 FIB rules), 1,500 ACLs
  - 16 Cisco routers
- Internet2
  - 100,000 IPv4 forwarding entries
  - 9 Juniper routers
## Test Packet Generation

<table>
<thead>
<tr>
<th></th>
<th>Stanford</th>
<th>Internet2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation Time</td>
<td>~1 hour</td>
<td>~40 min</td>
</tr>
<tr>
<td>Regular Packets</td>
<td>3,871</td>
<td>35,462</td>
</tr>
<tr>
<td>Packets/Port (Avg)</td>
<td>12.99</td>
<td>102.8</td>
</tr>
<tr>
<td>Min-Set-Cover Reduction</td>
<td>160x</td>
<td>85x</td>
</tr>
<tr>
<td>Ruleset structure</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

<1% Link Utilization when testing 10 times per second!
Using ATPG for Performance Testing

• Beyond functional problems, ATPG can also be used for detecting and localizing performance problems
• Intuition: generalize results of a test from success/failure to performance (e.g. latency)
• To evaluate used emulated Stanford Network in Mininet-HiFi
  – Open vSwitch as routers
  – Same topology, translated into OpenFlow rules
• Users can inject performance errors
Does it work?

• Production Deployment
  – 3 buildings on Stanford campus
  – 30+ Ethernet switches
    • Link cover only (instead of rule cover)
  – 51 test terminals
CS@Stanford Network Outage

Tue, Oct 2, 2012 at 7:54 PM:

“Between 18:20-19:00 tonight we experienced a complete network outage in the building when a loop was accidentally created by CSD-CF staff. We're investigating the exact circumstances to understand why this caused a problem, since automatic protections are supposed to be in place to prevent loops from disabling the network.”
The problem in the email

Unreported problem
ATPG Limitations

• Dynamic/Non-deterministic boxes
  – e.g. NAT
• “Invisible” rules
  – e.g. backup rules
• Transient network states
• Ambiguous states (work in progress)
  – e.g. ECMP
Related work

Policy
“Group X can talk to Group Y”

Forwarding Rules

Control Plane

Topology

Forwarding State

Forwarding Rule != Forwarding State

Topology on File != Actual Topology

NICE, Anteater
HSA, VeriFlow

ATPG
Takeaways

• ATPG tests the forwarding state by generating minimal link, queue, rule covers automatically
• Brings lens of testing and coverage to networks
• For Stanford/Internet2, testing 10 times per second needs <1% of link overhead
• Works in real networks.
Merci!

http://eastzone.github.com/atpg/