Automatic Test Packet Generation

Hongyi Zeng, Peyman Kazemian, Nick McKeown
University, Stanford, CA, USA

George Varghese
UCSD, La Jolla
Microsoft Research, Mountain View, CA, USA

https://github.com/eastzone/atpg/wiki
Debugging the net

Troubleshooting the network is difficult.
Troubleshooting the network is difficult.

- Survey of NANOG mailing list
  - Data set: 61 responders:
    - 23 medium size networks (<10K hosts),
      12 large networks (< 100K hosts).
  - Frequency: 35% generate >100 tickets per month.
  - Downtime: 56% take over an half hour to resolve.
  - Current tools: Ping (86%), Traceroute (80%).
  - 70% asked for better tools, automatic tests.
Automatic Test Packet Generation

Framework that test the liveness, congestion and performance.

Goal:
automatically generate test packets to test the network, and pinpoint faults.
ATPG use the header space framework.

Using the header space analyses we can collect the entire packet range from the net.
ATPG use the header space framework.

Reminder:

space represented by \( \{0,1\}^L \).

- Represent by port and header.
- Define how packet field are modified.
- Generate a list of output packets.
- Transfer function \( T \) - models a network.
ATPG use the header space framework.

- **Ordered List of Rules:** The packet match so far as it travers the net.

- **Topology:** Transfer function $\Gamma$ models the network by specifying which pairs of ports are connected.

- **Packet Life:** Applying the $T$ and $\Gamma$ repeatedly till the packet reach her destination.
Test packet generation

Assume there is a set of terminals in the network can send and receive test packets.

- ATPG use only available test terminals.
- ATPG use headers that test terminal is permitted to send.

Goal:

generate a set of test packets to exercise every rule.
Test packet generation

Test packet generator

Test Pkt DB

All pairs reachability

T function

parser

Topology, FIB, ALC ...

Test terminals

Header space analysis

Test packet generation
Test packet Selection

Determine the set of packet headers that can be sent from each terminal to another.

Generate all pairs reachability table, equivalent classes.

Sample and compression.

Sent periodically the test.
If a error is detected, use localization algorithm.

Complexity:
\[ O(TDR^2) \]

- T - number test terminals
- D - network diameter
- R - average number of Rules in box
Find all packet space
<table>
<thead>
<tr>
<th></th>
<th>Header</th>
<th>Ingress Port</th>
<th>Egress Port</th>
<th>Rule History</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>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>$p_2$</td>
<td>dst_ip = 10.2/16</td>
<td>$P_C$</td>
<td>$P_B$</td>
<td>$r_{C1}, r_{B1}, r_{A3}$ link CB, link BA</td>
</tr>
<tr>
<td>p1</td>
<td>dst_ip = 10.0/16, tcp = 80</td>
<td>PA</td>
<td>PB</td>
<td>( r_{A1}, r_{B3}, r_{B4} ) link AB</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------</td>
<td>----</td>
<td>----</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>p2</td>
<td>dst_ip = 10.2/16</td>
<td>PC</td>
<td>PB</td>
<td>( r_{C1}, r_{B1}, r_{A3} ) link CB, link BA</td>
</tr>
<tr>
<td>p3</td>
<td>dst_ip = 10.2/16</td>
<td>PB</td>
<td>PA</td>
<td>( r_{B1}, r_{A3} ) link AB</td>
</tr>
<tr>
<td>p4</td>
<td>dst_ip = 10.1/16</td>
<td>PB</td>
<td>PC</td>
<td>( r_{B2}, r_{C2} ) link BC</td>
</tr>
<tr>
<td>p5</td>
<td>dst_ip = 10.2/16</td>
<td>PC</td>
<td>PA</td>
<td>( r_{C1}, r_{A3} ) link BC</td>
</tr>
<tr>
<td>p6</td>
<td>dst_ip = 10.2/16, tcp = 80</td>
<td>PC</td>
<td>PB</td>
<td>( r_{C1}, r_{B3}, r_{B4} ) link BC</td>
</tr>
<tr>
<td>p7</td>
<td>dst_ip = 10.1/16</td>
<td>PA</td>
<td>PC</td>
<td>( r_{A2}, r_{C2} ) link AC</td>
</tr>
</tbody>
</table>

\( r_{A1} : \text{dst} \_ip = 10.0/16 \rightarrow \text{Port} \_0 \)
\( r_{A2} : \text{dst} \_ip = 10.1/16 \rightarrow \text{Port} \_1 \)
\( r_{A3} : \text{dst} \_ip = 10.2/16 \rightarrow \text{Port} \_2 \)

\( r_{B1} : \text{dst} \_ip = 10.2/16 \rightarrow \text{Port} \_2 \)
\( r_{B2} : \text{dst} \_ip = 10.1/16 \rightarrow \text{Port} \_1 \)
\( r_{B3} : \text{dst} \_ip = 10.0/16 \rightarrow r_{B4} \)
\( r_{B4} : \text{tcp} = 80 \rightarrow \text{Port} \_0 \)

\( r_{C1} : \text{In} \_Port = 1 \rightarrow \text{Port} : 0,2 \)
\( r_{C2} : \text{In} \_Port = 0,2 \rightarrow \text{Port} : 1 \)
Sampling

- Pick at least one packet from every equivalent class.
- You can peak randomly or peak all the headers from a certain class to check a fault for a specific header.

<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>$r_{A1}, r_{B3}, r_{B4}$ link AB</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$P_C$</td>
<td>$P_B$</td>
<td>$r_{C1}, r_{B1}, r_{A3}$ link CB, link BA</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$P_B$</td>
<td>$P_A$</td>
<td>$r_{B1}, r_{A3}$ link AB</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$P_B$</td>
<td>$P_C$</td>
<td>$r_{B2}, r_{C2}$ link BC</td>
</tr>
<tr>
<td>$p_5$</td>
<td>$P_C$</td>
<td>$P_A$</td>
<td>$r_{C1}, r_{A3}$ link BC</td>
</tr>
<tr>
<td>$p_6$</td>
<td>$P_C$</td>
<td>$P_B$</td>
<td>$r_{C1}, r_{B3}, r_{B4}$ link BC</td>
</tr>
<tr>
<td>$p_7$</td>
<td>$P_A$</td>
<td>$P_C$</td>
<td>$r_{A2}, r_{C2}$ link AC</td>
</tr>
</tbody>
</table>
Compression

- Using the Min-Set Cover algorithm.
- The remaining test packet will be used later for fault localization.

<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>$r_{A1}, r_{B3}, r_{B4}$ link AB</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$P_C$</td>
<td>$P_B$</td>
<td>$r_{C1}, r_{B1}, r_{A3}$ link CB, link BA</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$P_B$</td>
<td>$P_A$</td>
<td>$r_{B1}, r_{A3}$ link AB</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$P_B$</td>
<td>$P_C$</td>
<td>$r_{B2}, r_{C2}$ link BC</td>
</tr>
<tr>
<td>$p_5$</td>
<td>$P_C$</td>
<td>$P_A$</td>
<td>$r_{C1}, r_{A3}$ link BC</td>
</tr>
<tr>
<td>($p_6$)</td>
<td>$P_C$</td>
<td>$P_B$</td>
<td>$r_{C1}, r_{B3}, r_{B4}$ link BC</td>
</tr>
<tr>
<td>($p_7$)</td>
<td>$P_A$</td>
<td>$P_C$</td>
<td>$r_{A2}, r_{C2}$ link AC</td>
</tr>
</tbody>
</table>
Fault Localization

- Success and failure depend on the nature of the rule forwarding / drop.

- Divide the fault in two categories:
  - Action fault - effect all the packets matching the rule.
  - Match fault - effect some of the packets matching the rule.
Find the set of suspect rules

- $P$ is the set of all the rules traversed by **success** test packets.
- $F$ is the set of all the rules traversed by **failing** test packets.
- $F - P$ is the set of suspect rules.

Minimize the suspect rules set

- We will make use of our reserved packets.

<table>
<thead>
<tr>
<th>Header</th>
<th>Ingress Port</th>
<th>Egress Port</th>
<th>Rule History</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$ dst_ip = 10.0/16, tcp = 80</td>
<td>$P_A$</td>
<td>$P_B$</td>
<td>$r_{A1}, r_{B3}$ link AB</td>
<td>pass</td>
</tr>
<tr>
<td>$p_2$ dst_ip = 10.2/16</td>
<td>$P_C$</td>
<td>$P_B$</td>
<td>$r_{C1}, r_{B1}$ link CB</td>
<td>fail</td>
</tr>
<tr>
<td>$p_3$ dst_ip = 10.2/16</td>
<td>$P_B$</td>
<td>$P_A$</td>
<td>$r_{B1}, r_{A3}$ link AB</td>
<td>fail</td>
</tr>
</tbody>
</table>
False positive

- There is a possibly for false positive, Rule that are left as suspect although it behave correctly.

- Can be solve by adding more test terminals to the network.
All pairs reachability

Test packet generator

Fault Localization

Test terminals

Test Pkt

DB

Header space analysis

Topology, FIB, ALC…

T function

parser
Use case for ATPG

- Test functional correctness of a network.
  - Forward rule and Drop rule:
    - if packet exercises the rule.
  - Link rule:
    - if packet passing the link without header modification.

- Test performance of the network.
  - Congestion and available bandwidth:
    - Measure the latency, bandwidth.
Drop rule
Implementation

- ATPG written in python.
- The data plane information collected and parsed (Cisco IOS).
- The generator uses Hassel header space analysis library to construct T functions.
- All pairs reachability is computed using multiprocess parallel processing.
- After Min set cover executes, the output is stored in SQLite database.
- The process uses JSON to communicate with the test agents.
Alternate implementation & SDN

In SDN, the controller could directly instruct the switch’s to send test packets and to forward test packet back to the controller.
Testing in Emulated Network

Replicated the Stanford backbone network in Mininet.

Using open vSwitch, real port configuration and real topology.
Testing in Emulated Network
Forwarding Error

Change forwarding rule to drop rule in boza.

$R_{boza}^1 : \text{dst\_ip} = 172.20.10.32/27 \rightarrow \text{drop}$
Forwarding Error

$R_1^{boza} : \text{dst}_ip = 172.20.10.32/27 \rightarrow \text{drop}$

<table>
<thead>
<tr>
<th>Header</th>
<th>Ingress</th>
<th>Egress</th>
<th>Rule History</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{dst}_ip = 172.20.10.33$</td>
<td>goza</td>
<td>coza</td>
<td>$[R_1^{goza}, L_{goza}^S, S_1, L_{bbra}^S, R_1^{bbra}, L_{bbra}^S, S_4, R_1^{coza}]$</td>
<td>Pass</td>
</tr>
<tr>
<td>$\text{dst}_ip = 172.20.10.33$</td>
<td>boza</td>
<td>coza</td>
<td>$[R_1^{boza}, L_{boza}^S, S_5, L_{bbra}^S, R_1^{bbra}, L_{bbra}^S, S_4, R_1^{coza}]$</td>
<td>Fail</td>
</tr>
<tr>
<td>$\text{dst}_ip = 171.67.222.65$</td>
<td>boza</td>
<td>poza</td>
<td>$[R_2^{boza}, L_{boza}^S, S_5, L_{bbra}^S, R_2^{bbra}, L_{bbra}^S, S_2, R_2^{poza}]$</td>
<td>Pass</td>
</tr>
</tbody>
</table>
Congestion

- bbra
- s1
- pozЬ
- poza
- yoza
- roza

![Graph showing throughput and latency over time with lines for pozЬ-roza and pozЬ-yoza]
<table>
<thead>
<tr>
<th></th>
<th>Ingress Port</th>
<th>Egress Port</th>
<th>Rule History</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>poza</td>
<td>yoza</td>
<td>Poza-s1,s1-bbra,bbra-yoza</td>
</tr>
<tr>
<td>$p_2$</td>
<td>roza</td>
<td>yoza</td>
<td>Roza-s2,s2-bbra,bbra-yoza</td>
</tr>
<tr>
<td>$p_3$</td>
<td>pzob</td>
<td>yoza</td>
<td>Pozb-s1,s1-bbra,bbra-yoza</td>
</tr>
<tr>
<td>$p_4$</td>
<td>pzob</td>
<td>roza</td>
<td>Pozb-s1,s1-bbra,bbra-roza</td>
</tr>
</tbody>
</table>
Production Network Deployment

Deployed an experimental ATPG system in Stanford University.

For test terminal they use 53 Wi-Fi access.

Test agent download the test packets from a central web server every ten minutes, and conducts a ping test every 10 seconds.
Production Network Deployment

During the experiment, a major outage occurred, ATPG has captured the problem.
Evaluation

Evaluated the system on Stanford backbone.

<table>
<thead>
<tr>
<th>Stanford (298 ports)</th>
<th>10%</th>
<th>40%</th>
<th>70%</th>
<th>100%</th>
<th>Edge (81%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Packets</td>
<td>10,042</td>
<td>104,236</td>
<td>413,158</td>
<td>621,402</td>
<td>438,686</td>
</tr>
<tr>
<td>Regular Packets</td>
<td>725</td>
<td>2,613</td>
<td>3,627</td>
<td>3,871</td>
<td>3,319</td>
</tr>
<tr>
<td>Coverage</td>
<td>22.2%</td>
<td>57.7%</td>
<td>81.4%</td>
<td>100%</td>
<td>78.5%</td>
</tr>
<tr>
<td>Computation Time</td>
<td>152.53s</td>
<td>603.02s</td>
<td>2,363.67s</td>
<td>3,524.62s</td>
<td>2,807.01s</td>
</tr>
</tbody>
</table>

testing all rules 10 times per second needs < 1% of link overhead.
Limitation

Dynamic, Non deterministic boxes
- Internal state can be changed.
- Random to route packets.

Sampling time
- ATPG uses sampling to collect the forwarding state.

Invisible rule
- Backup rules changes may be occurred without been notice.
Conclusion

- ATPG tests liveness and performance.
- < 1% of link overhead.
- Works in real networks.

Questions?