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

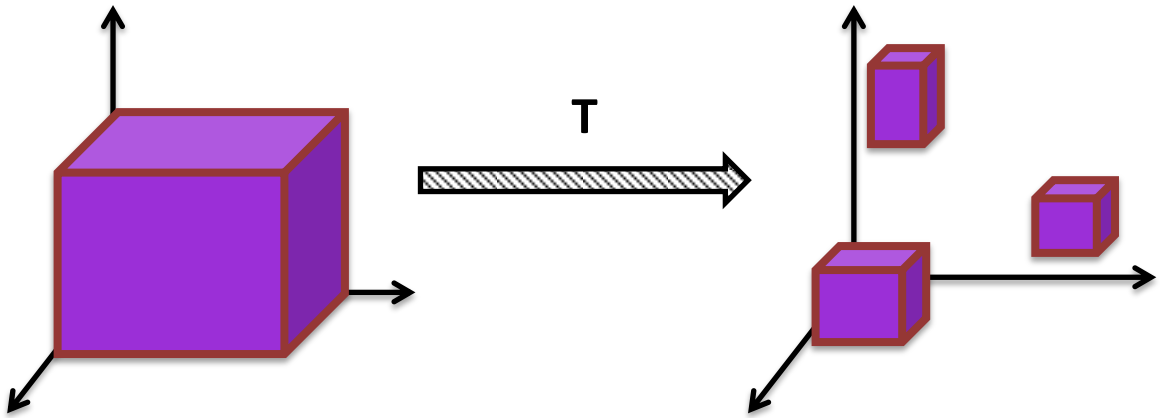
Framework that test the liveness , congestion and performance.

Goal:

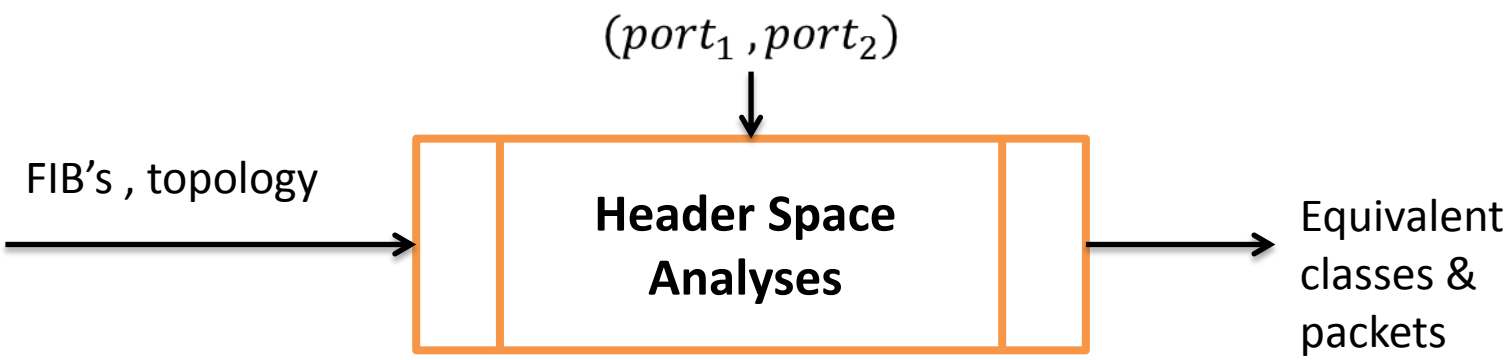
automatically generate test packets to test the network,

and pinpoint faults.

ATPG use the header space framework.



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

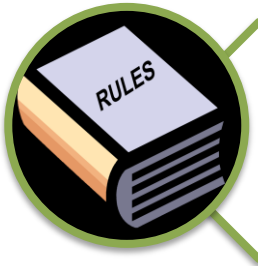


ATPG use the header space framework.

Reminder :

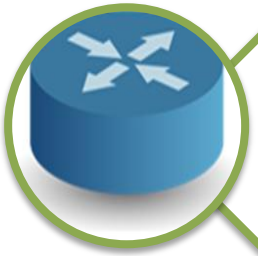
space *represnt by* $\{0,1\}^L$.

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 Γ models the network by specifying which pairs of ports are connected.

Packet Life

applying the T and Γ repeatedly till the packet reach her destination.

Test packet generation

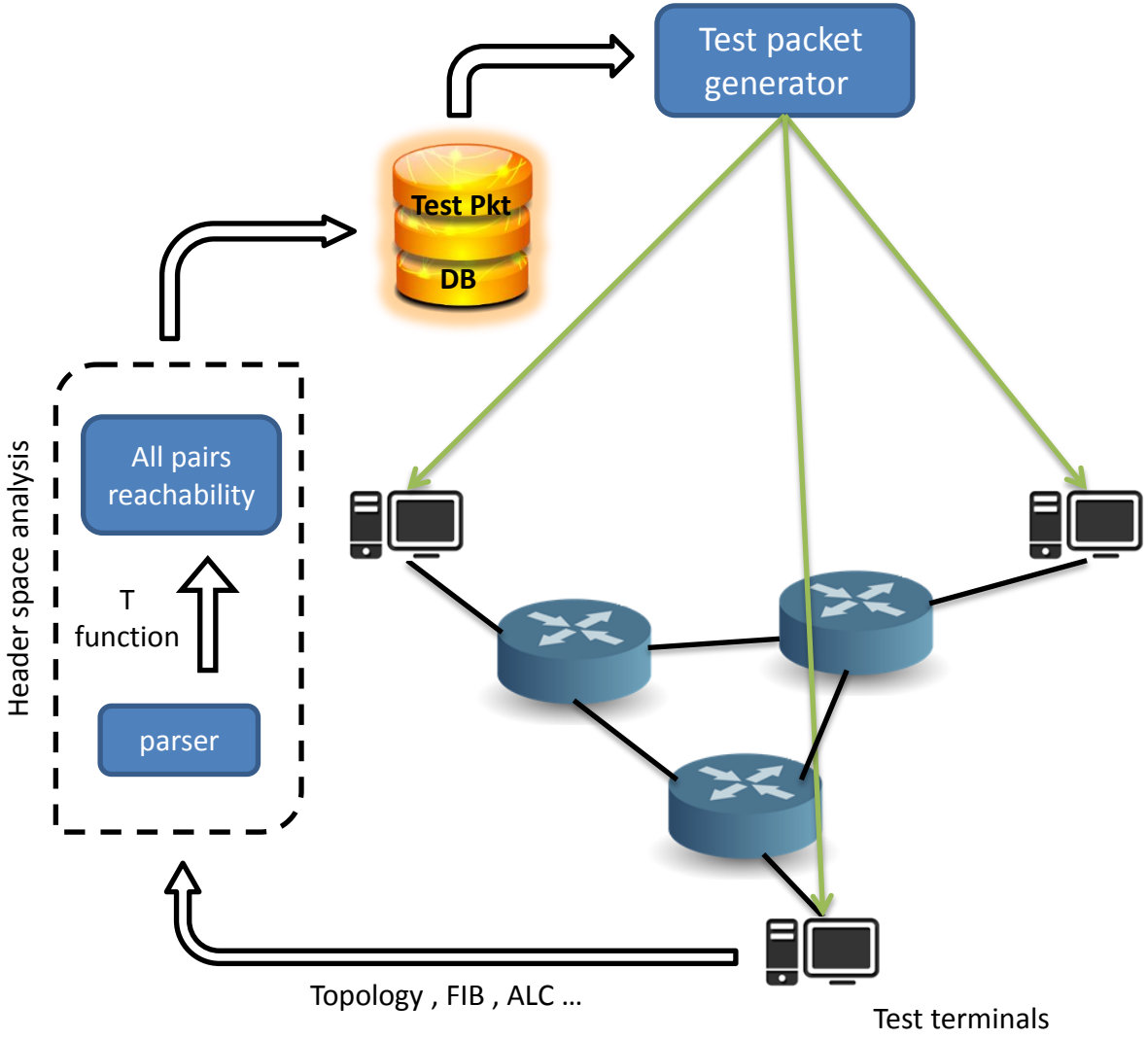
Assume there is a set of terminals in the network can send and receive test packets.

- 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 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.

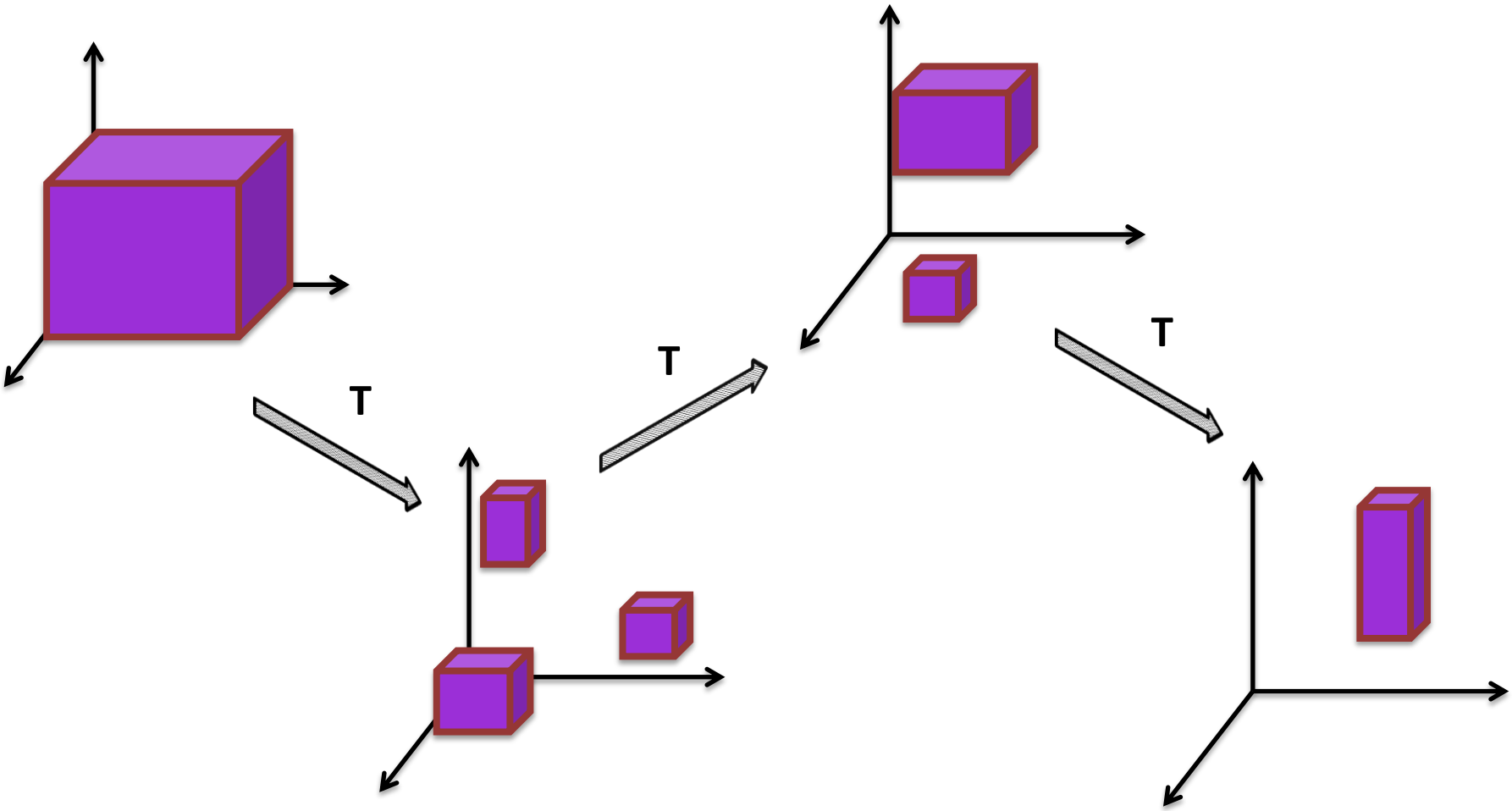
Sent periodically the test .

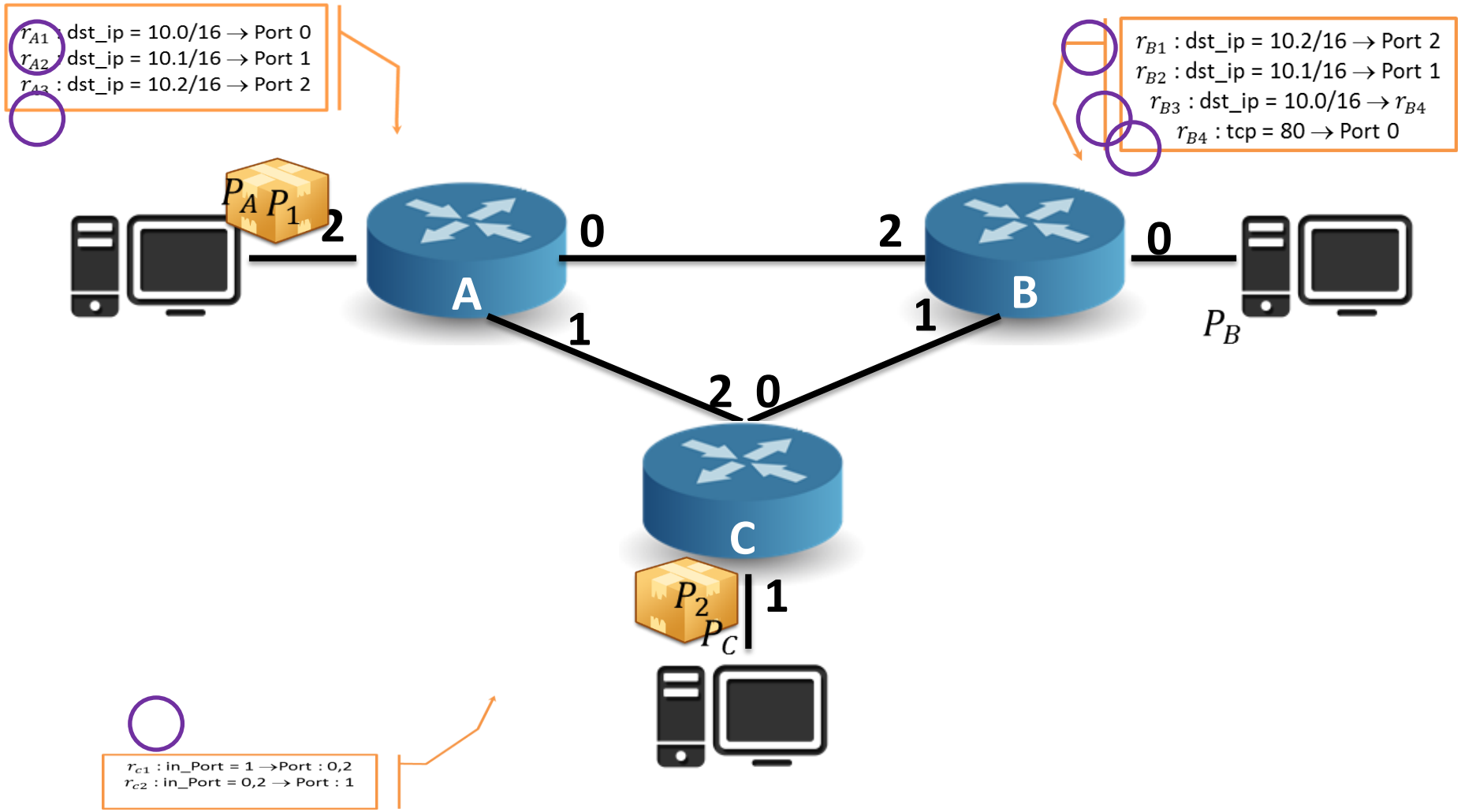
If a error is detected , use localization algorithm.

Complexitiy $O(TDR^2)$

T - number test terminals
D - network diameter
R - average number of Rules in box

Find all packet space





	Header	Ingress Port	Egress Port	Rule History
p_1	dst_ip = 10.0/16, tcp = 80	P_A	P_B	r_{A1}, r_{B3}, r_{B4} link AB
p_2	dst_ip = 10.2/16	P_C	P_B	r_{c1} , r_{B1}, r_{A3} link CB,link BA

r_{A1} : dst_ip = 10.0/16 → Port 0
r_{A2} : dst_ip = 10.1/16 → Port 1
r_{A3} : dst_ip = 10.2/16 → Port 2

r_{B1} : dst_ip = 10.2/16 → Port 2
r_{B2} : dst_ip = 10.1/16 → Port 1
r_{B3} : dst_ip = 10.0/16 → r_{B4}
r_{B4} : tcp = 80 → Port 0

	Header	Ingress Port	Egress Port	Rule History
p_1	dst_ip = 10.0/16, tcp = 80	P_A	P_B	r_{A1}, r_{B3}, r_{B4} link AB
p_2	dst_ip = 10.2/16	P_C	P_B	r_{C1}, r_{B1}, r_{A3} link CB , link BA
p_3	dst_ip = 10.2/16	P_B	P_A	r_{B1}, r_{A3} link AB
p_4	dst_ip = 10.1/16	P_B	P_C	r_{B2}, r_{C2} link BC
p_5	dst_ip = 10.2/16	P_C	P_A	r_{C1}, r_{A3} link BC
p_6	dst_ip = 10.2/16, tcp = 80	P_C	P_B	r_{C1}, r_{B3}, r_{B4} link BC
p_7	dst_ip = 10.1/16	P_A	P_C	r_{A2}, r_{c2} link AC

r_{c1} : in_Port = 1 →Port : 0,2
r_{c2} : in_Port = 0,2 → 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.

	Header	Ingress Port	Egress Port	Rule History
p_1	dst_ip = 10.0/16, tcp = 80	P_A	P_B	r_{A1}, r_{B3}, r_{B4} link AB
p_2	dst_ip = 10.2/16	P_C	P_B	r_{C1}, r_{B1}, r_{A3} link CB , link BA
p_3	dst_ip = 10.2/16	P_B	P_A	r_{B1}, r_{A3} link AB
p_4	dst_ip = 10.1/16	P_B	P_C	r_{B2}, r_{C2} link BC
p_5	dst_ip = 10.2/16	P_C	P_A	r_{C1}, r_{A3} link BC
p_6	dst_ip = 10.2/16, tcp = 80	P_C	P_B	r_{C1}, r_{B3}, r_{B4} link BC
p_7	dst_ip = 10.1/16	P_A	P_C	r_{A2}, r_{c2} link AC

Compression

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

	Header	Ingress Port	Egress Port	Rule History
p_1	dst_ip = 10.0/16, tcp = 80	P_A	P_B	r_{A1}, r_{B3}, r_{B4} link AB
p_2	dst_ip = 10.2/16	P_C	P_B	r_{C1}, r_{B1}, r_{A3} link CB , link BA
p_3	dst_ip = 10.2/16	P_B	P_A	r_{B1}, r_{A3} link AB
p_4	dst_ip = 10.1/16	P_B	P_C	r_{B2}, r_{C2} link BC
p_5	dst_ip = 10.2/16	P_C	P_A	r_{C1}, r_{A3} link BC
(p_6)	dst_ip = 10.2/16, tcp = 80	P_C	P_B	r_{C1}, r_{B3}, r_{B4} link BC
(p_7)	dst_ip = 10.1/16	P_A	P_C	r_{A2}, r_{c2} link AC

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.

Fault Localization

- 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.

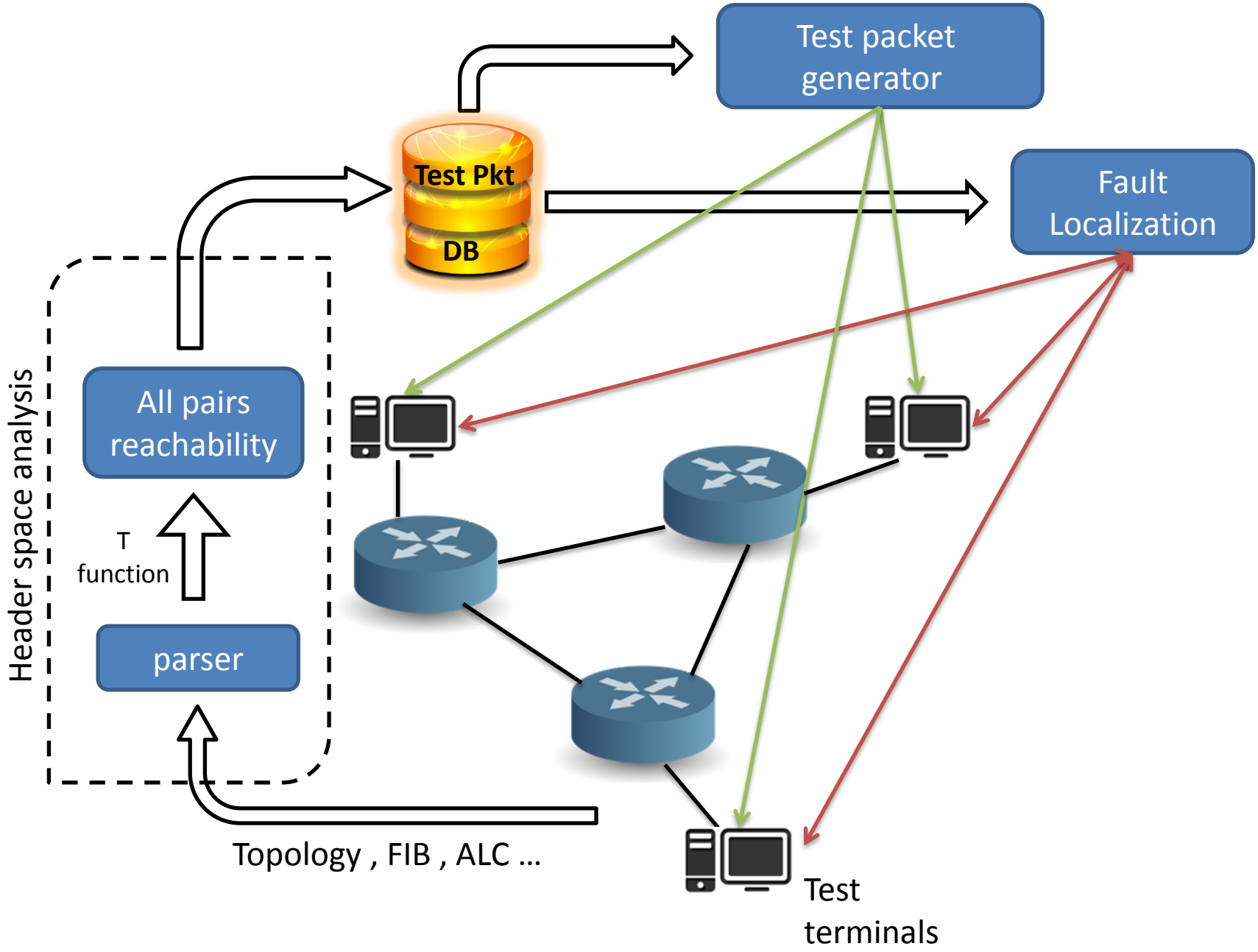
- Minimize the suspect rules set
  - We will make use of our reserved packets.

	Header	Ingress Port	Egress Port	Rule History	Result
p_1	dst_ip = 10.0/16, tcp = 80	P_A	P_B	r_{A1}, r_{B3} link AB	pass
p_2	dst_ip = 10.2/16	P_C	P_B	r_{C1}, r_{B1} link CB	fail
p_3	dst_ip = 10.2/16	P_B	P_A	r_{B1}, r_{A3} link AB	fail

False positive

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

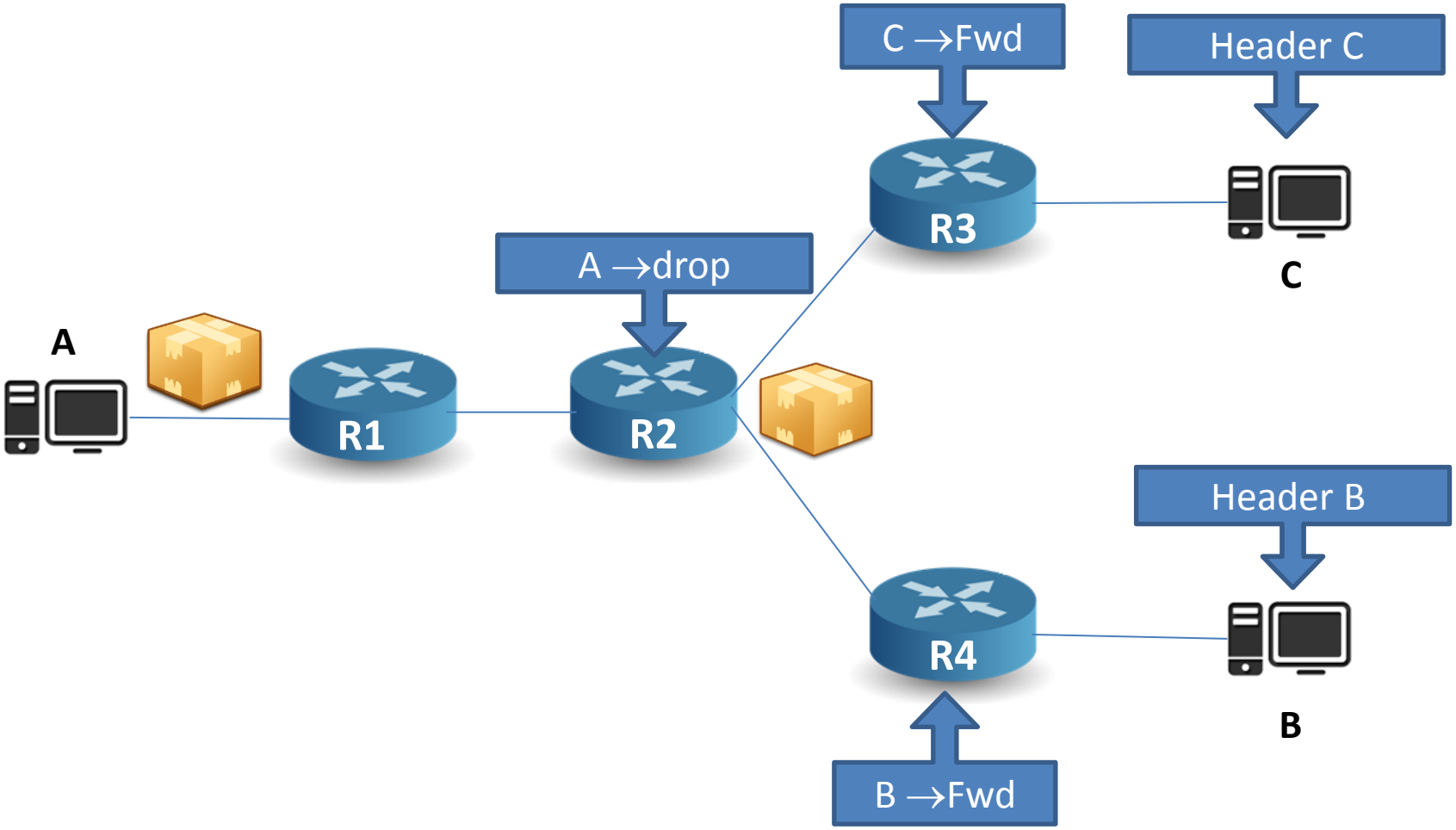
- Can be solve by adding more test terminals to the network.


Test packet generator
Test Pkt
DB
Fault Localization
Header space analysis
All pairs reachability
T function
parser
Topology , FIB , ALC ...
Test terminals

Use case for ATPG

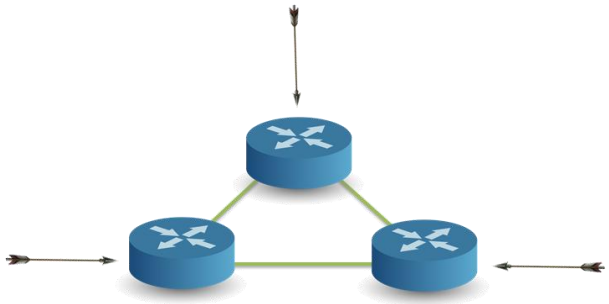
- Test functional correctness of a network.
  - Forward rule and Drop rule :
    - if packet exercises the rule.
  - Link rule :
    - if packet passing the link without header modification.
- Test performance of the network.
  - Congestion and available bandwidth:
    - Measure the latency , bandwidth.

Drop rule



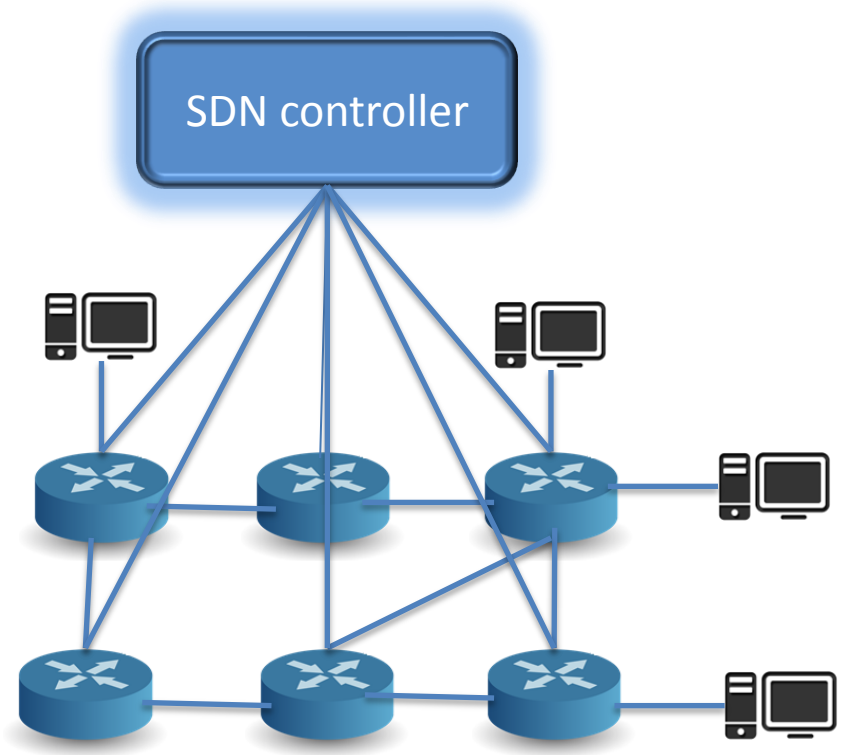
Implementation

- ATPG written in python.
- The data plane information Collected and parsed (Cisco IOS) .
- The generator use 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 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.



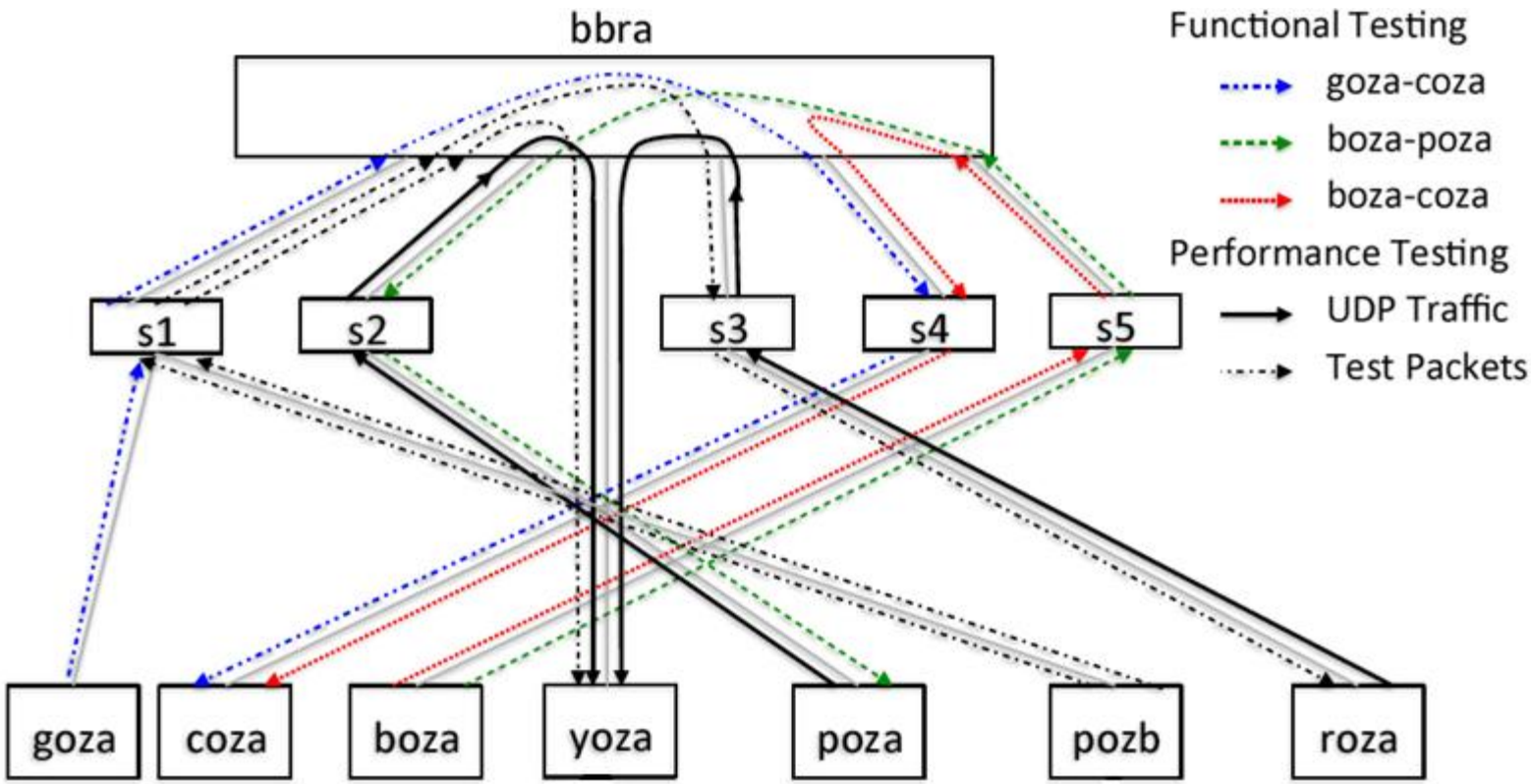
Testing in Emulated Network

Replicated the Stanford backbone network in Mininet.

Using open vSwitch ,

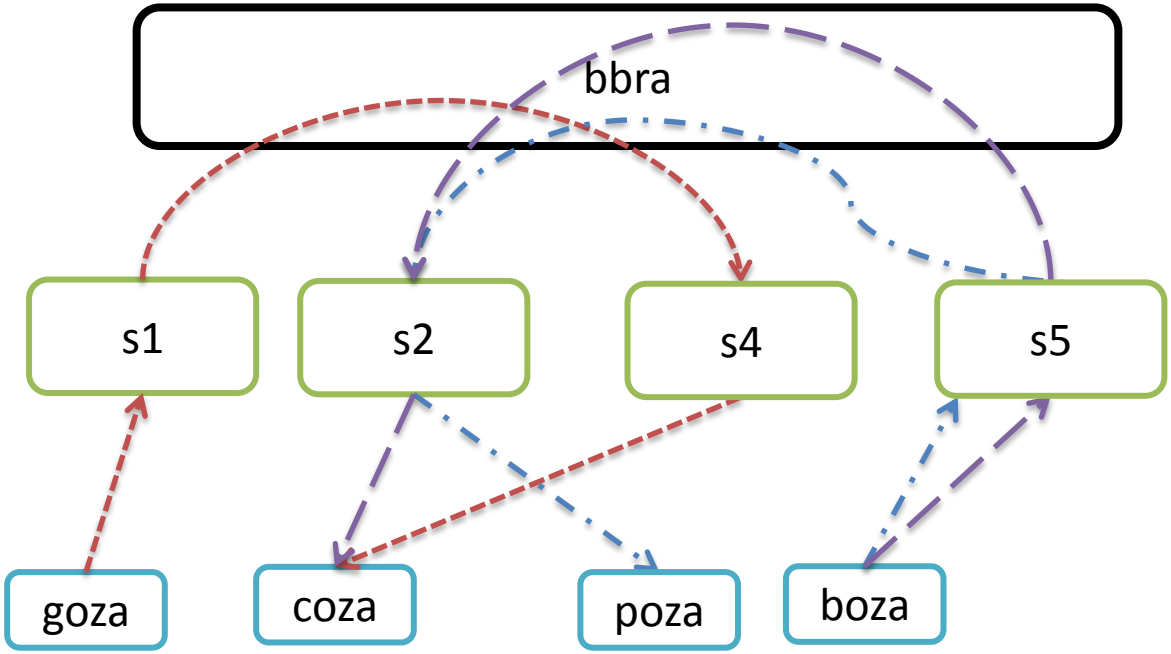
real port configuration and real topology.

Testing in Emulated Network



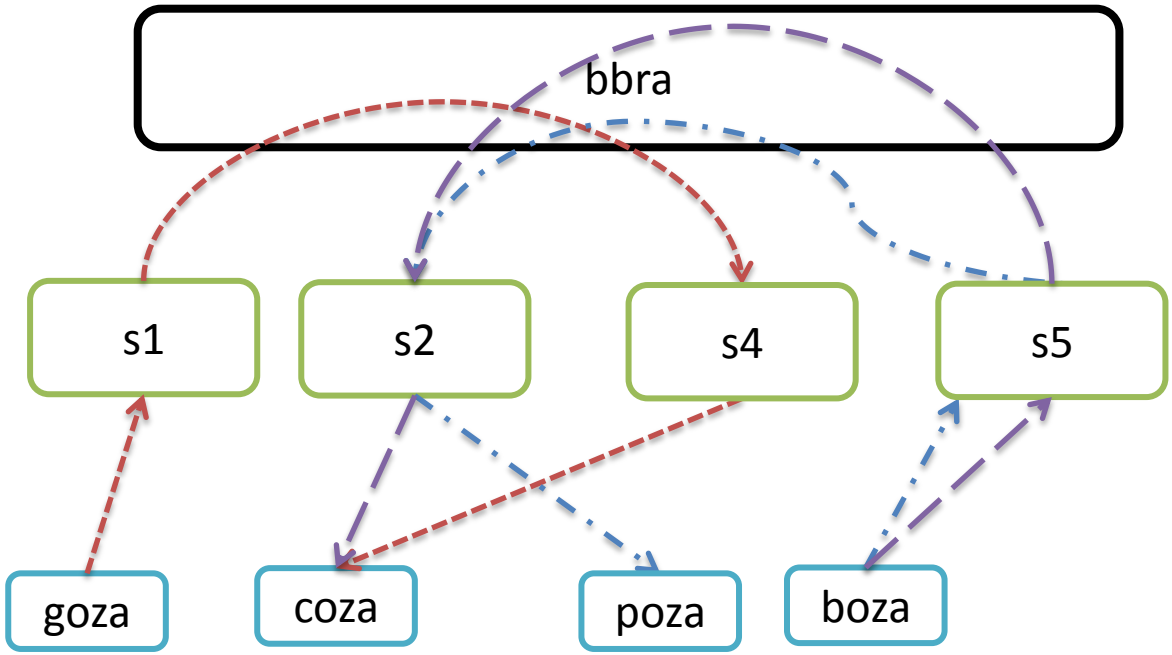
Forwarding Error

- Change forwarding rule to drop rule in **boza**.
  - R_1^{boza} : dst_ip = 172.20.10.32/27 → *drop*



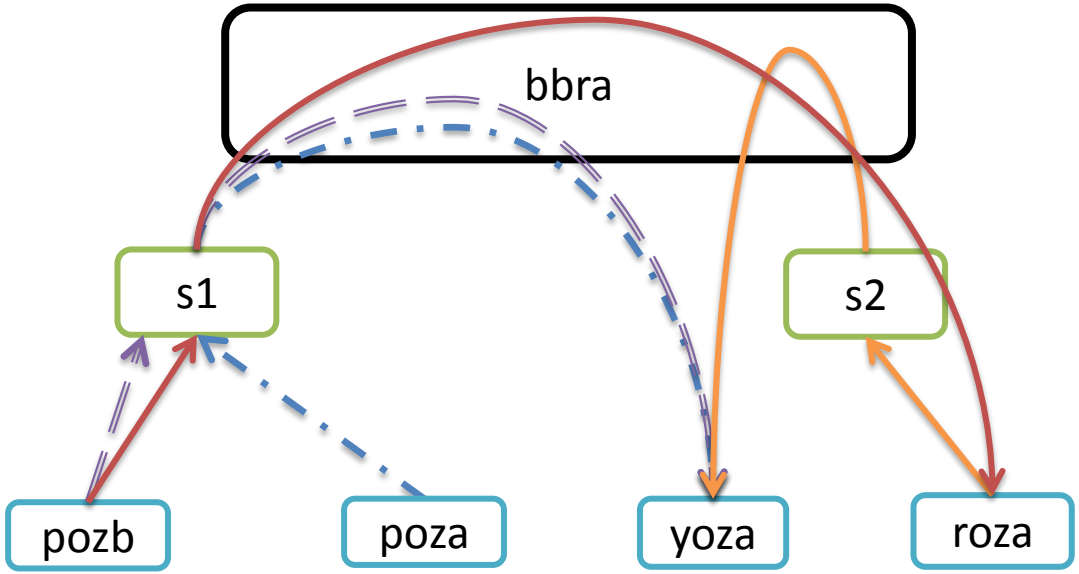
Forwarding Error

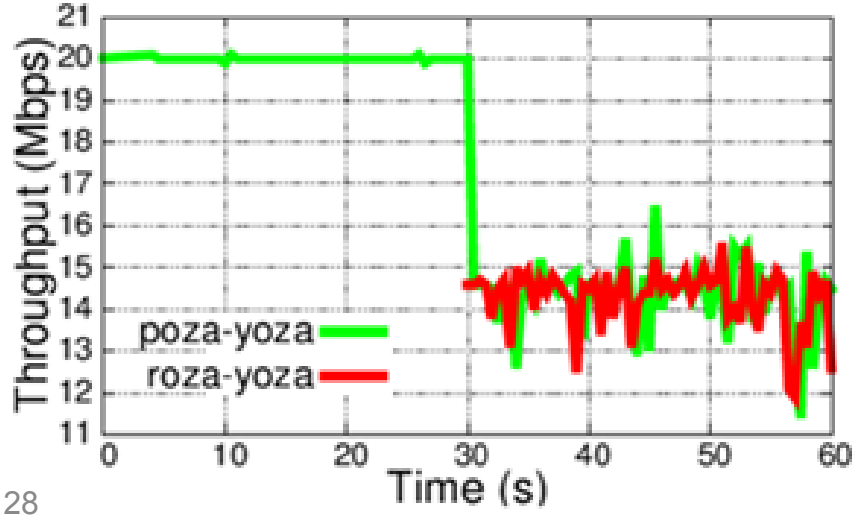
$\curlywedge R_1^{boza}$: dst_ip = 172.20.10.32/27 → *drop*

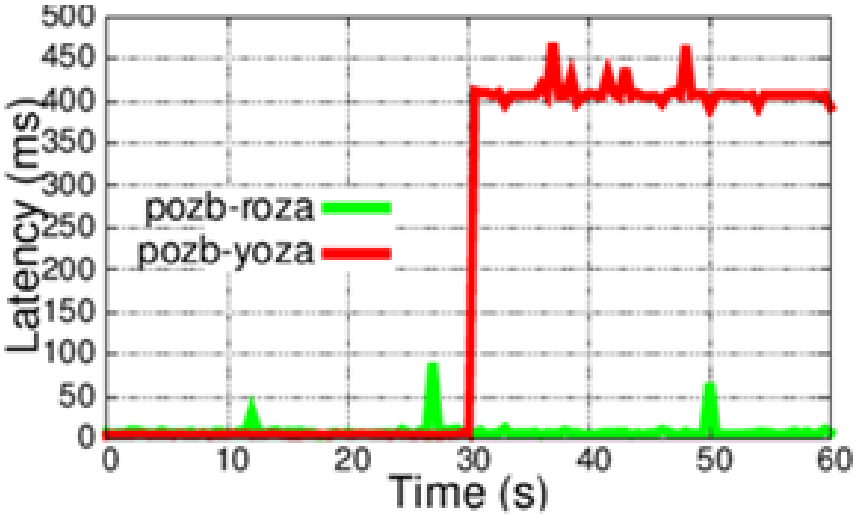


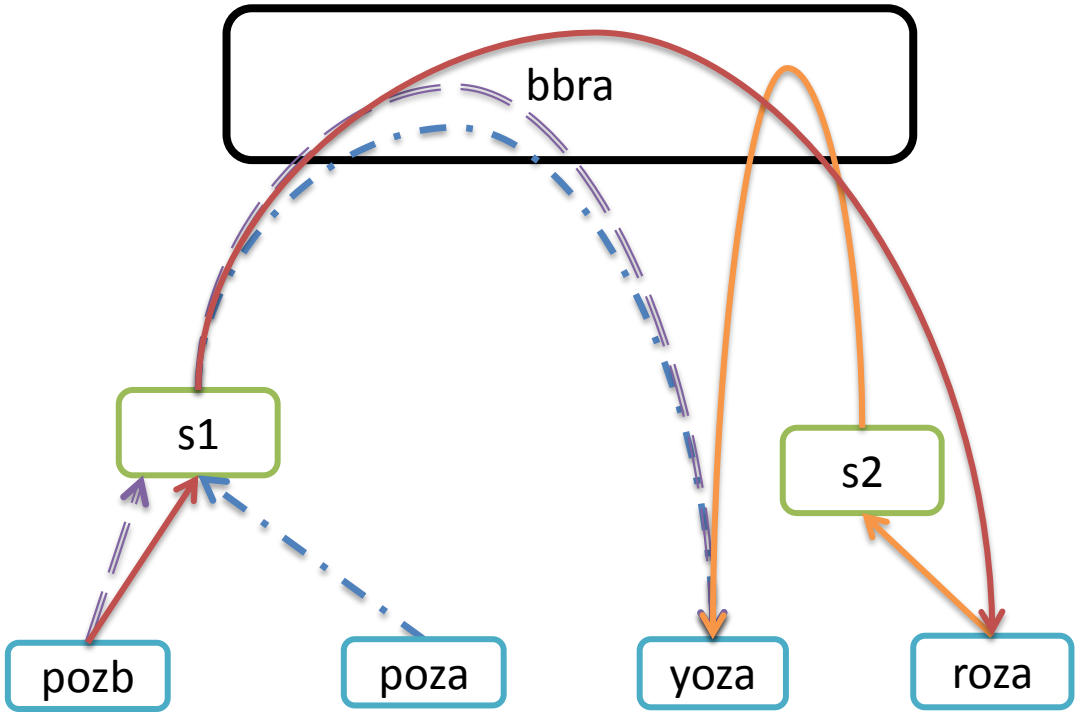
Header	Ingress	Egress	Rule History	Result
$dst\_ip = 172.20.10.33$	*goza*	*coza*	$[R_1^{goza}, L_{goza}^{S_1}, S_1, L_{S_1}^{bbra}, R_1^{bbra}, L_{bbra}^{S_4}, S_4, R_1^{coza}]$	Pass
$dst\_ip = 172.20.10.33$	*boza*	*coza*	$[R_1^{boza}, L_{boza}^{S_5}, S_5, L_{S_5}^{bbra}, R_1^{bbra}, L_{bbra}^{S_4}, S_4, R_1^{coza}]$	Fail
$dst\_ip = 171.67.222.65$	*boza*	*poza*	$[R_2^{boza}, L_{boza}^{S_5}, S_5, L_{S_5}^{bbra}, R_2^{bbra}, L_{bbra}^{S_2}, S_2, R_2^{poza}]$	Pass

Congestion


bbra
s1
s2
pozb
poza
yoza
roza


Throughput (Mbps)
Time (s)
poza-yoza
roza-yoza


Latency (ms)
Time (s)
pozb-roza
pozb-yoza



	Ingress Port	Egress Port	Rule History
p_1	poza	yoza	Poza-s1,s1-bbra,bbra-yoza
p_2	roza	yoza	Roza-s2,s2-bbra,bbra-yoza
p_3	pzob	yoza	Pozb-s1,s1-bbra,bbra-yoza
p_4	pzob	roza	Pozb-s1,s1-bbra,bbra-roza

Production Network Deployment

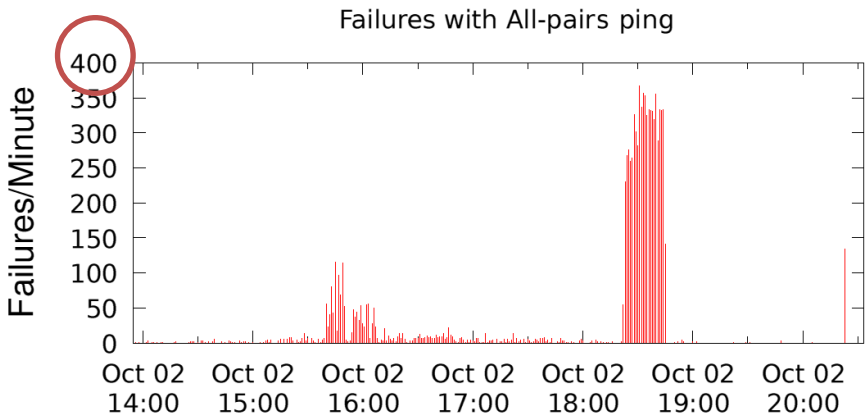
Deployed an experimental ATPG system in Stanford University.

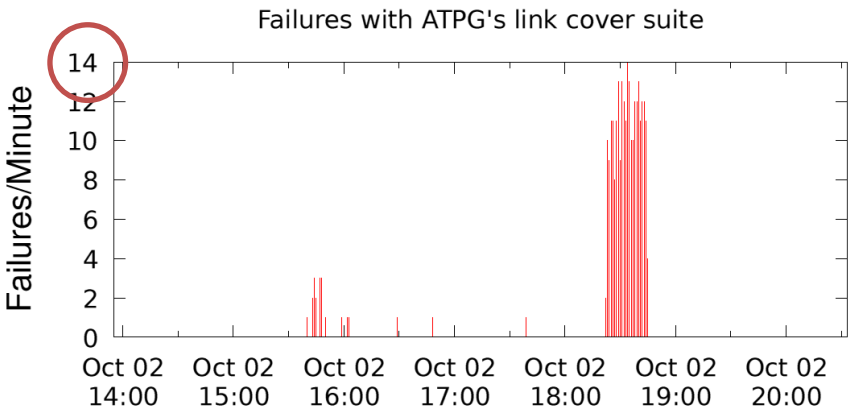
For test terminal they use 53 Wi-Fi access.

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.

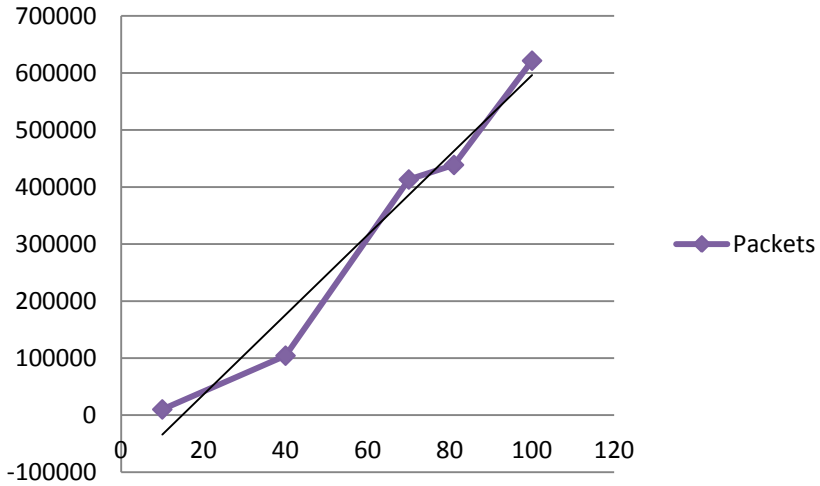


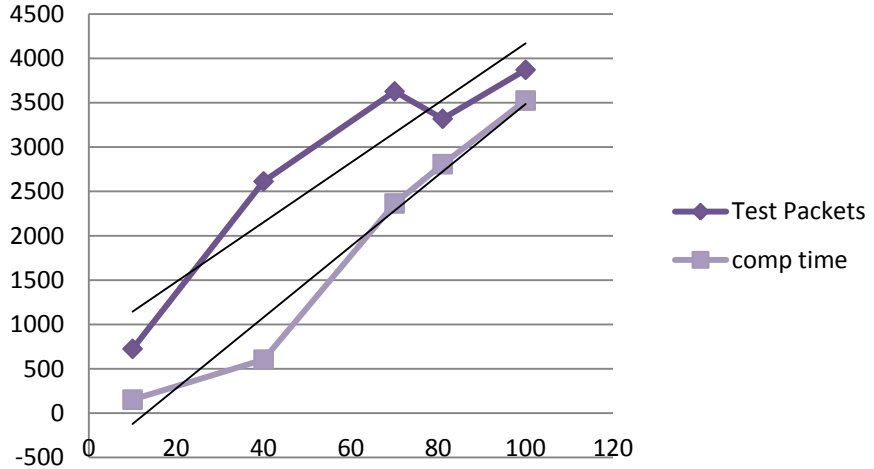


Evaluation

Evaluated the system on Stanford backbone.

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





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?