Test packet generation automatic way in a network

Balakrishna.G, Rajesh.Y
Computer Department, Andhra Loyola inst of eng & technology,Vijayawada

***Abstract* - We propose an automated and systematic approach for testing and debugging networks called "Automatic Test Packet Generation" (ATPG). ATPG reads router configurations and generates a device-independent model. The model is used to generate a minimum set of test packets to (minimally) exercise every link in the network or (maximally) exercise every rule in the network. Test packets are sent periodically, and detected failures trigger a separate mechanism to localize the fault. ATPG can detect both functional and performance problems.**

***Keywords* - Data plane analysis, network troubleshooting, test packet generation**

1.INTRODUCTION

It is notoriously hard to debug networks. Every day, network engineers wrestle with router misconfigurations, fiber cuts, faulty interfaces, mislabeled cables, software bugs, intermittent links, and a myriad other reasons that cause net- works to misbehave or fail completely. We tested our method on two real-world data sets—the back- bone networks of Stanford University, Stanford, CA,USA, and Internet2. Representing an enterprise network and a nationwide ISP.

The results are encouraging: Thanks to the structure of real world rule sets, the number of test packets needed is surprisingly small. For the Stanford network with over 757000 rules and more than 100 VLANs, we only need 4000 packets to exercise all forwarding rules and ACLs. On Internet2, 35 000 packets suffice to exercise all IPv4 forwarding rules. Put another way, we can check every rule in every router on the Stanford backbone 10 times every second by sending test packets that consume less than 1% of network bandwidth. The link cover for Stanford is even smaller, around 50 packets, which allows proactive livens testing every millisecond using 1% of net- work bandwidth.

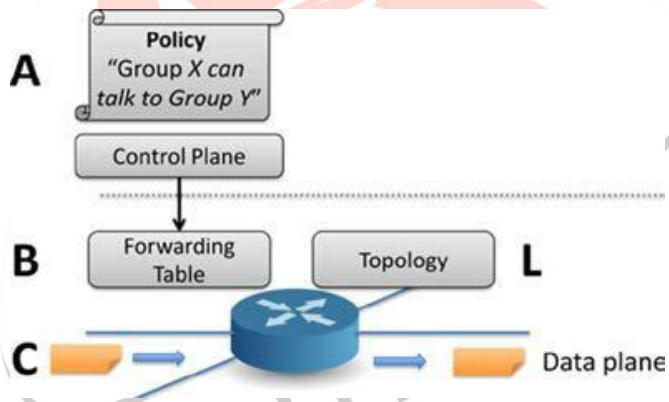

Fig .1 Static versus dynamic checking

The contributions of this paper are as follows:

1. a survey of network operators revealing common failures and root causes;
2. a test packet generation algorithm ;
3. a fault localization algorithm to isolate faulty devices and rules ;
4. ATPG use cases for functional and performance testing ;

2. EXISTING SYSTEM

Testing likeness of a network is a fundamental problem for ISPs and large data center operators. Sending probes between every pair of edge ports is neither exhaustive nor scalable . It suffices to find a minimal set of end-to-end packets that traverse each link. However, doing this requires a way of abstracting across device specific configuration files, generating headers and the links they reach and finally determining a minimum set of test packets (Min-Set-Cover). To check enforcing consistency between policy and the configurative.

Disadvantages of Existing System

- Not designed to identify livens failures, bugs router hardware or software, or performance problems.
- The two most common causes of network failure are hardware failures and software bugs.
- That problems manifest themselves both as reach ability failures and throughput/latency degradation.

3. PROPOSED SYSTEM

Framework that automatically generates a minimal set of packets to test the livens of the underlying topology and the congruence between data plane state and configuration specifications. The tool can also automatically generate packets to test performance assertions such as packet latency. It can also be specialized to generate a minimal set of packets that merely test every link for network livens

Advantages of Proposed System:

- A survey of network operators revealing common failures and root causes.
- A test packet generation algorithm.
- A fault localization algorithm to isolate faulty devices and rules.
- ATPG use cases for functional and performance testing.

Evaluation of a prototype ATPG system using rule sets collected from the Stanford and Internet2 backbones.

3.1. Current Proposal

ATPG uses the header space framework—a geometric model of how packets are processed. In header space, protocol-specific meanings associated with headers are ignored: A header is viewed as a flat sequence of ones and zeros. A header is a point (and a flow is a region) in the space, where is an upper bound on header length. By using the header space framework, we obtain a unified, vendor-independent, and protocol-agnostic model of the network2 that simplifies the packet generation process significantly.

A. Definitions:

Summarizes the definitions in our model.

Packets: A packet is defined by a(port, header) tuple, where the denotes a packet's position in the network at any time instant; each physical port in the network is assigned a unique number.

Switches: A switch transfer function, T, models a network device, such as a switch or router. Each network device contains a set of forwarding rules (e.g., the forwarding table) that determine how packets are processed. An arriving packet is associated with exactly one rule by matching it against each rule in descending order of priority, and is dropped if no rule matches.

Rules: A rule generates a list of one or more output packets, corresponding to the output port(s) to which the packet is sent, and defines how packet fields are modified. The rule abstraction models all real-world rules we know including IP forwarding (modifies port, checksum, and TTL, but not IP address); VLAN tagging (adds VLAN IDs to the header); and ACLs (block a header, or map to a queue). Essentially, a rule defines how a region of header space at the ingress (the set of packets matching the rule) is transformed into regions of header space at the egress.

Rule History: At any point, each packet has a rule history: an ordered list[r0,r1...] of rules the packet matched so far as it traversed the network. Rule histories are fundamental to ATPG, as they provide the basic raw material from which ATPG constructs tests.

Topology: The topology transfer function, , models the network topology by specifying which pairs of ports(Psrc,Pdst) are

Bit	b= 0I1Ix
Header	h =[b0,b1,b2….bL]
Port	p=1I2I...INIdrop
Packet	pk = (p, h)
Rule	r : pk —> pK or [pk]
Match	r.matchset : [pk]
Transfer Function	T : pK —>pK or [pK]
Topo Function	T: (Psrc, h)-> (pdst, h)

```
function Ti(pk)
        #Iterate according to priority in switch i
for r ∈ ruleseti do
        if pk ∈ r.tnatchset then
                pk.history <-- pk.historyU{r}
                return r(pk)
        return {(drop,pk.h)}


function NETWORK(packets, switches, T')
for pk0 ∈ packets do
        T <—FIND_swiTcH(pko.p, switches)
for pk1 ∈ T(pko) do
        if pk1.p ∈ EdgePorts then
        #Reached edge
        RECORD(pk1)
else
        #Find next hop
```

NETWORK(T(pk1), switches, T)

Life of a packet: repeating and until the packet reaches its destination or is dropped. connected by links. Links are rules that forward packets from Psrc to Pdst to without modification. If no topology rules match an input port, the port is an edge port, and the packet has reached its destination.

B. Life of a Packet

The life of a packet can be viewed as applying the switch and topology transfer functions repeatedly. When a packet pK arrives at a network port , the switch function that contains the input port pK.p is applied to pK, producing a list of new packets[pK1,pK2,...]. If the packet reaches its destination, it is recorded. Otherwise, the topology function is used to invoke the switch function containing the new port. The process repeats until packets reach their destinations (or are dropped).

4.IMPLEMENTATION

Java Technology:

Java technology is both a programming language and a platform.

The Java Programming Language:

The Java programming language is a high level language that can be characterized by all of the following buzzwords:

- Simple
- Dynamic
- Object oriented
- Portable
- Distributed High performance

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called Java byte codes —the platform independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.

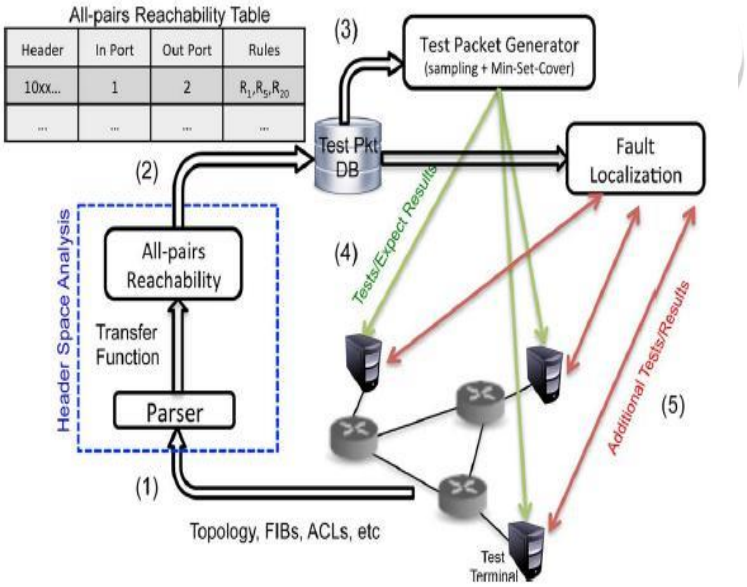

Fig 4.1.Automatic Test Packet

5.IMPLIMENATION

Modules:

- Test Packet Generation
- Generate All-Pairs Reach ability Table
- ATPG Tool
- Fault Localization

Modules Description:

Test Packet Generation: We assume a set of test terminals in the network can send and receive test packets. Our goal is to generate a set of test packets to exercise every rule in every switch function, so that any fault will be observed by at least one test packet. This is analogous to software test suites that try to test every possible branch in a program. The broader goal can be limited to testing every link or every queue. When generating test packets, ATPG must respect two key constraints First Port

(ATPG must only use test terminals that are available) and Header (ATPG must only use headers that each test terminal is permitted to send).

Generate All-Pairs Reach ability Table: ATPG starts by computing the complete set of packet headers that can be sent from each test terminal to every other test terminal. For each such header, ATPG finds the complete set of rules it exercises along the path. To do so, ATPG applies the all-pairs reach ability algorithm described. On every terminal port, an all- header (a header that has all wild carded bits) is applied to the transfer function of the first switch connected to each test terminal. Header constraints are applied here.

ATPG Tool: ATPG generates the minimal number of test packets so that every forwarding rule in the network is exercised and covered by at least one test packet. When an error is detected, ATPG uses a fault localization algorithm to determine the failing rules or links.

Fault Localization: ATPG periodically sends a set of test packets. If test packets fail, ATPG pinpoints the fault(s) that caused the problem. A rule fails if its observed behavior differs from its expected behavior. ATPG keeps track of where rules fail using a result function "Success" and "failure" depend on the nature of the rule: A forwarding rule fails if a test packet is not delivered to the intended output port, whereas a drop rule behaves correctly when packets are dropped. Similarly, a link failure is a failure of a forwarding rule in the topology function. On the other hand, if an output link is congested, failure is captured by the latency of a test packet going above a threshold.

6.CONCLUSION

Network managers today use primitive tools such as ping and trace route. Our survey results indicate that they are eager for more sophisticated tools. Other fields of engineering indicate that these desires are not unreasonable: For example, both the ASIC and software design industries are buttressed by billion- dollar tool businesses that supply techniques for both static (e.g., design rule) and dynamic (e.g., timing) verification. In fact, many months after we built and named our system, we discovered to our surprise that ATPG was a well-known acronym in hardware chip testing, where it stands for Automatic Test Pattern Generation [2]. We hope network ATPG will be equally useful for automated dynamic testing of production networks.

Testing livens of a network is a fundamental problem for ISPs and large data center operators. Sending probes between every pair of edge ports is neither exhaustive nor scalable. It suffices to find a minimal set of end-to-end packets that traverse each link. However, doing this requires a way of abstracting across device specific configuration files (e.g., header space), generating headers and the links they reach (e.g., all-pairs reach ability), and finally determining a minimum set of test packets . Even the fundamental problem of automatically generating test packets for efficient livens testing requires techniques akin to ATPG.

References

[1] "ATPG code repository," [Online]. Available: http://eastzone.github.com/atpg/

[2] "Automatic Test Pattern Generation," 2013 [Online]. Available: http://en.wikipedia.org/wiki/Automatic_test_pattern_generation

[3] P. Barford, N. Duffield, A. Ron, and J. Sommers, "Network performance anomaly detection and localization," in Proc. IEEE INFOCOM, Apr. , pp. 1377–1385.

[4] "Beacon," [Online]. Available: http://www.beaconcontroller.net/

[5] Y. Bejerano and R. Rastogi, "Robust monitoring of link delays and faults in IP networks," IEEE/ACM Trans. Netw., vol. 14, no. 5, pp. 1092–1103, Oct. 2006.

[6] C. Cadar, D. Dunbar, and D. Engler, "Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs," in Proc. OSDI, Berkeley, CA, USA, 2008, pp. 209–224.