Volume 2, Issue 12, December-2015, pp. 1173-1177 ISSN (O): 2349-7084

International Journal of Computer Engineering In Research Trends

Available online at: www.ijcert.org

Automatic and Systematic Methodology for Test Packet Generation

[1]Prasanna, [2] CH.Harika

[1](M.Tech) CSE, Dept. of Computer Science and Engineering

[2]Assistant Professor, Dept. of Computer Science and Engineering

Priyadarshini Institute of Technology & Science

Abstract: As of late systems are becoming wide and more complex. In any case, managers use devices like ping and trace route to investigate issues. Consequently, we proposed a programmed and Systematic methodology for testing and investigating systems called Automated Test Packet Generation (ATPG). This methodology gets switch designs and creates a device autonomous model. ATPG create a couple set of Packet Generation to discover each connection in the system. Test bundles are sent as often as possible and it recognize disappointments to limit the issue. ATPG can distinguish both practical and execution (throughput, inertness) issues. We found, less number of Packet Generation is sufficient to test all principles in systems. For case, 4000 parcels can cover all tenets in Stanford spine system while 53 are sufficiently much to cover all connections.

Keywords: Fault Localization, Test Packet Selection, Network Debugging, Automatic Test packet Generation (ATPG).

1. INTRODUCTION

It is popularly known us, very difficult to troubleshoot or identify and remove errors in networks. Every day, network engineers fight with mislabeled cables, software bugs, router misconfigurations, fiber cuts, faulty interfaces and other reasons that cause networks to drop down. Network engineers hunt down bugs with various tools (e.g., Ping, trace route, SNMP) and track down the reason for network failure using a combination of accrued wisdom and impression. Debugging networks is becoming more harder as networks are growing larger (modern data centers may contain 10 000 switches, a campus network may serve 50 000 users, a 100-Gb/s long-haul link may carry 100 000 flows) and are getting complicated (with over 6000 RFCs, router software was based on millions of lines of source code, and network chips contain billions of gates. Fig. 1 is a simplified view of network state. Bottom of the figure is the forwarding state to forward each packet, consist of L2 and L3 forwarding information base (FIB), access control lists, etc. The forwarding state was written by the control plane (that could be local or remote) and should correctly implement the network administrator's scheme. Examples of the scheme include: "Security group X was isolated from security Group Y," "Use OSPF for routing," and "Video traffic received at least 1 Mb/s." We could think of the controller compiling the scheme (A) into device specific configuration files (B), which in turn determine the forwarding behavior of each packet (C). To ensure the network behave as designed, the three steps should remain consistent every times. Minimally, requires that sufficient links and nodes are working; the control plane identifies that a laptop can access a server, the required outcome can fail if links fail. The main reason for network failure is hardware and software failure, and this problem is recognized themselves as reach ability failures and throughput/latency degradation.ATPG detects errors independently and exhaustively testing forwarding entries and packet processing rules in network. In this tool, test packets are created algorithmically from the device configuration files and First information base, with minimum number of packets needed for complete coverage. Test packets are fed into the network in which every rule was exercised directly from the data plan. Since ATPG treats links just like normal forwarding rules, the full coverage provides testing of every link in network. It could be particularized to generate a minimal set of packets that test every link for network liveness. For reacting to failures, many network operators like Internet proactively test the health of the network by pinging between all pairs of sources. Organizations can modify ATPG to face their needs; for example, they can select to test for network liveness (link cover) or test every rule (rule cover) to make sure security policy. ATPG could be modified to test reach ability and performance. ATPG can adapt to constraints such as taking test packets from only a few places in the network or using particular routers to generate test packets from every port.

II. NETWORK MODEL

To send and receive test packets, network monitor assumes special test agents in the network. The network monitor gets the database and builds test packets and instructs each agent to send the proper packets. Recently, test agents partition test packets by IP Proto field and TCP/UDP port number, but other fields like IP option can be used. If any tests fail, the monitor chooses extra test packets from booked packets to find the problem. The process gets repeated till the fault has been identified. To communicate with test agents, monitor uses JSON, and SQLite's string matching to lookup test packets efficiently ATPG uses the header space framework—a geometric model of how packets are processed we described in [16] (and used in [31]). 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. 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

III. RELATED WORK

The test packets which generate automatically by configuration is not aware by earlier techniques. The very often related works we are familiar is offline tools which test invariants in networks. In control plane, NICE [7] tries to comprehensively cover code path symbolically in a controller applications with support of simplified switch and host models. In the data plane, Anteater [25] models invariants as a Boolean satisfiability problem which tests them against configurations with a SAT solver. Header Space Analysis [16] use geometric model for checking reach ability, detecting loops, and for verifying slicing. Recently, SOFT [1] put forward to check uniformity between different Open Flow agent implementations which is responsible for bridging control and data planes in SDN context. ATPG supplement these checkers directly by verifying the data plane and exercising a important set of dynamic or performance errors which could not be captured. The major contribution of ATPG is not fault localization, but deciding a compact set of end-to-end measurements which could exercise every rule and every link. The mapping in between Min-Set-Cover and network monitoring was been explored previously in [3] and [5]. ATPG progress the detection granularity to rule level by working router configuration and data plane information. ATPG not limited to liveness testing, but it can be applicable for checking higher level properties like performance. Our work was closely related to work in programming languages and symbolic debugging. We made a preliminary tries to use KLEE [6] and find it to be 10 times slower than the unoptimized header space framework. We speculate this is basically because in our framework we directly simulate the forward path of a packet in addition of solving constraints using an SMT solver. However, more work is needed to understand the differences and potential opportunities. Connected by links. Links are rules that forward packets from 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 (Fig. 4). When a packet arrives at a network port , the switch function that contains the input port is applied to , producing a list of new packets . 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).

IV. EXISTING SYSTEM

Testing liveness 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 configuration.

1. Packets
A packet is taken by number of the denotes a packet's places in the network at any time and locations Instant; every physical node in the network is assigned a same id number.

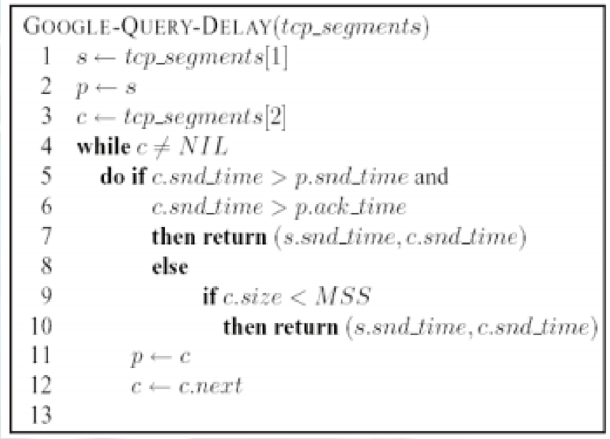
2. Switches A switch transfer function models is taken a network device such as a switch Each network device contains a set of forwarding conditions that determine packets are processed An arriving packet is associated with different rule by matching it against each rule in descending order in different locations and is dropped if no rule matches in different models.

3. Rules A rule generates a list of one or more output packets corresponding to the output port to which the packet is sent and defines how packet fields are changed The rule abstraction models all real world conditions we know including IP forwarding and ACLs Essentially a rule defines how a region of header space at the ingress is transformed into regions of header space at the egress

4. Rule History Take any node each packet has conditions history and ordered list of rules the packet matched and traveling the network. Rule histories are fundamental problems provided provide the basic raw material

5. Topology The topology different function models the network models by specifying which pairs of ports and links are rules that forward packets from to without changing. If no models rules match an input port the port is an edge port and the packet has reached its destination

A. Test Packet Generation Algorithm We take a set of test nodes in the network model send and receive test packets Our aim is to generate a set of test packets and changes every conditions in every switch objects so that every fault will be observed by at least one test packet This is scanner type software test models that try to test every possible branch models The broader goal can be limited to testing every link every queue When generating test packets ATPG must respect two key constraints **(1) Port:** ATPG must be test terminals that are available **(2) Header:** ATPG must be use headers that each test terminal is permitted to send For the network administrator may only allow using a specific model of VLAN



```
GOOGLE-QUERY-DELAY(tcp_segments)
s ← tcp_segments[1]
p ← s
c ← tcp_segments[2]
while c ≠ NIL
   do if c.snd_time > p.snd_time and
         c.snd_time > p.ack_time
         then return (s.snd_time, c.snd_time)
         else
              if c.size < MSS
                 then return (s.snd_time, c.snd_time)
       p ← c
       c ← c.next
```

Fig. 2: Segments Methods Test packets are taken into the network in which that every rule is covered directly from the data plane with different locations treats links like normal forwarding conditions its full coverage provides testing of every link in the network model It can also be specialized to form a minimal no of of packets that obviously test every link for network likeness At least one basic form we would feel that ATPG similar technique is fundamental to networks Instead of reacting to failures many network operators such as Internet2 proactively check the health of their network using pings between all pairs of sources all-pairs of network provide testing of all links and has been found for large networks model such as Planet Lab.

V. PROPOSED SYSTEM

Contender framework generates minimum no of packets automatically to debug the false occurring in the network model This tool could automatically generate packets for checking performance assertions such as like packet loss finds and determines errors by independently testing all forwarding entries any packet processing rules and security models in network test packets are generated algorithmically from device configuration files and from FIBs which requires minimum number of packets for complete coverage Test packets are fed into the network in which that every rule is covered directly from the data plane Since treats links like normal forwarding conditions its full coverage provides testing of every link in the network model It can also best specialized to form a minimal set of packets that obviously test every link for network likeness At least in this basic form, we would feel that some different technique is fundamental to networks Instead of reacting to failures many network operators such as proactively check the health of their network using pings between all pairs of sources all-pairs does not provide testing of all links and has been found to be unsalable for large networks such as Planet Lab

VI. METHODOLOGY

The proposed system can be divided into following modules: 1. Failures and root causes of network operators 2. Data plane analysis 3. Network troubleshooting 4. ATPG system 5. Network Monitor

1. Failure and Root Causes of Network Operators Network traffic is represented to a specific queue in router but these packets are drizzled because the rate of token bucket low It is difficult to troubleshoot a network for three different models First the forwarding state is shared to multiple routers and security and is determined by the forwarding data filter conditions and configuration parameters Second the forwarding state is difficult to watch because it requires manually logging into every box in the network model Third the forwarding state is edited simultaneously by different programs protocols and humans.

2. Data Plane Analysis Automatic Test Packet Generation framework which automatically generates a minimum set of packets to check the likeness of underlying network models and congruence different data plane state and configuration specifications These model can automatically generate packets to test performance assertions like packet latency ATPG find faults by independently and exhaustively checking all security rules forwarding entries and packet processing conditions in network. The test packets are generated algorithmically from the device configuration different files and FIBs, with less number of packets needed for whole coverage Test packets are fed in the network so that every rule is covered directly from the data plane This tool can be customized to check only for reach ability or for its performance

3. Network Troubleshooting
The cost of network debugging is captured by two metrics one is the number of network-related tickets per month and another is the average time taken to resolve a ticket there are 35% of networks which generate more than 100 tickets per month. Of the respondents, 40.4% estimate takes under 30 minutes to resolve a ticket if asked what the ideal tool for network debugging it is would be, 70.7% reports automatic test generation to check performance and correctness. Some of them added a desire for long running tests to find jitter or intermittent real-time link capacity monitoring and monitoring tools for network state. In short, while our survey is small, it helps the hypothesis that network administrators face complicated symptoms and causes.

4. ATPG Systems
Depending on network model ATPG generates less number of test packets so that every forwarding rule is exercised and covered by at least one test packet when an error is found, ATPG use different localization algorithm to ascertain the failing rules in network model

5. Network Monitor

To send and receive test data packet network monitor assumes special test agents in the network The network monitor gets the database and builds test packets and instructs each different to send the proper packets Recently test agents partition test packets by IP Proto field and TCP/UDP port number but other fields like IP option can be used If any tests fail the monitor chooses extra test packets from booked packets to find the faults The process gets repeated till the fault has been identified To communicate with test agents monitor uses and SQL it string matching to lookup test packets efficiently

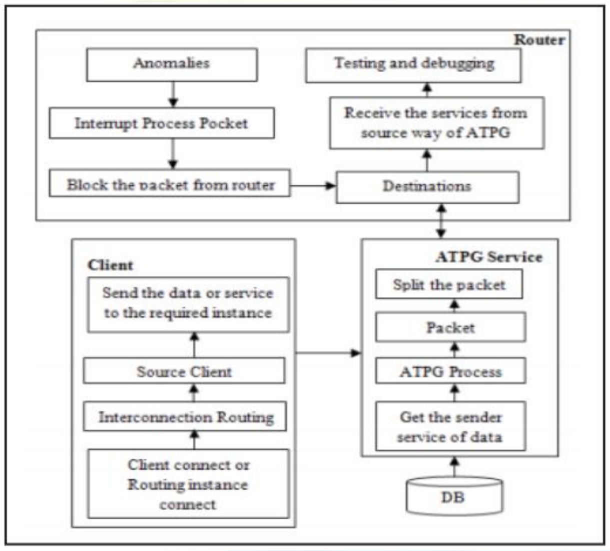

Router
Anomalies
Testing and debugging
Interrupt Process Pocket
Receive the services from source way of ATPG
Block the packet from router
Destinations
Client
Send the data or service to the required instance
Source Client
Interconnection Routing
Client connect or Routing instance connect
ATPG Service
Split the packet
Packet
ATPG Process
Get the sender service of data
DB

Fig. 2. ATPG system block diagram

VII. CONCLUSION

In current System it uses a method that is neither exhaustive nor scalable different it reaches all pairs of edge nodes it could take detect faults in likeness properties ATPG goes much further than likeness testing with different framework ATPG could test for reach ability model and performance methods Our implementation also enlarges testing with simple errors localization scheme also build using header space framework

REFERENCES

[1] Hongyi Zeng, Peyman Kazemian, George Varghese, ACM, and Nick McKeown, ACM,“Automatic Test Packet Generation”.

[2] M. Jain, C. Dovrolis,“End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput”, IEEE/ACM Trans. Netw., Vol. 11, No. 4, pp. 537–549, Aug. 2003.

[3] Kompella, R. R., Greenberg, A., Rexford, J., Snoeren, A. C., Yates, J. Cross-layer Visibility as a Service", In Proc. of fourth workshop on Hot Topics in Networks (HotNet-IV) (2005)

[4] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, A. Greenberg,"Routing design in operational networks: A look from the inside", In Proc. ACM SIGCOMM, 2004.

[5] Mark Stemm, Randy Katz, Srinivasan Seshan,“A network measurement architecture for adaptive applications”, In Proceedings of the nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 285 - 294, 2000.

[6] Verma, D.,"Simplifying Network Administration using Policy based Management".

[7] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, S.-J. Lee, “S3: A scalable sensing service for monitoring large networked systems".

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest,"Introduction To Algorithms", Eastern Economy Edition.New Delhi: Prentice-Hall of India Private Limited 1998.

[9] E Balagurusamy,"Programming in ANSI C", Second edition. New Delhi: Tata McGraw-Hill, 1997.

[10] Deitel & Deitel, C++ How To Program, Second Edition, New Jersey, NJ Prentice Hall, 1997

[11] Tom Christiansen, Nathn Torkington,"Perl Cookbook: Solutions and Examples For Perl Programmers", 1st Edition, Sebastopol, CA: O’Reilly & Associates, Inc., 1999.

[12] Nathan Patwardhan, Ellen Siever and Stephen Spainhour, PERL In A Nutshell, 2nd Edition, Sebastopol, CA: O’Reilly & Associates, Inc., 2002.

[13] Tom Milligan, “Using Perl With Rational ClearCase Automation Library”, 2004Retrieved Nov10, 2010 from