International Journal of Combined Research & Development (IJCRD)
eISSN: 2321-225X;pISSN:2321-2241 Volume: 4; Issue: 10; October -2015

Automatic Assessment Packet Generation & Fault Localization System

Jagadish Math[1], Sameena Banu[2]

[1]*P.G.Student, Department of Computer Science & Engineering,*
Khaja Bandanawaz College of Engineering& Technology,Kalburgi,Karntaka,India.
[2]*Assistant Professor, Department of Computer Science & Engineering,*
Khaja Bandanawaz College of Engineering&Technology,Kalburgi,Karntaka,India.

***Abstract*— Recently networks are growing wide and more complex. However administrators use tools like ping and trace route to debug problems. Hence we proposed an automatic and Methodical approach for testing and debugging networks called Automatic Test Packet Generation (ATPG). This approach gets router configurations and generates a device-independent model. ATPG generate a few set of test packets to find every link in the network. Test packets are forwarded frequently and it detect failures to localize the fault. ATPG can detect both functional and performance (throughput, latency) problems. We found, less number of test packets is enough to test all rules in networks. For example, 4000 packets can cover all rules in Stanford backbone network, while 53 are much enough to cover all links.**

Keywords— **Fault Localization, Test Packet Selection, Network Debugging, Automatic Test packet Generation (ATPG), Forwarding InformationBase (FIB).**

I. 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 devicespecific 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 reachability failures and throughput/latency degradation. Our intention is to automatically find these kinds of failures.
The intention of this paper is to generate a minimum set of packets automatically to cover every link in the network.
This tool can automatically generate packets to test performance assertions like packet latency. 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 reachability 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.

The contributions of this paper are as follows:

1) A survey of network operators exposing common failures and root causes.
2) A test packet generation algorithm.
3) A fault localization algorithm to separate faulty devices and Rules.
4) ATPG usecases for functional and throughput testing.
5) Evaluation of prototype ATPG system using rule sets gathered from the Stanford and Internet2 backbones.

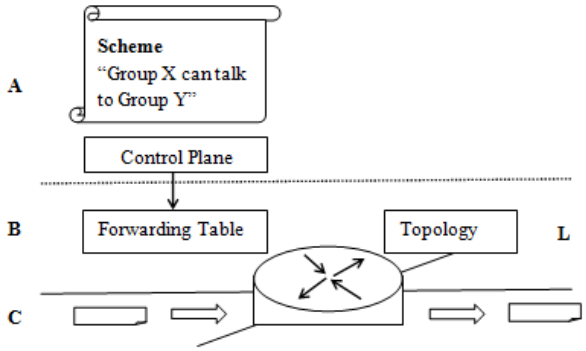

Fig. 1. Static versus dynamic checking: A scheme is compiled to forwarding state, and it is executed by the forwarding plane.

II. 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 reachability, 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.

III. PROBLEM DEFINITION

In current system, the administrator manually decides which ping packet to be sent. Sending programs between every pair of edge ports is neither extensive nor scalable. This system is enough to find minimum set of end-to-end packets that travel each link. However, doing this need a way of abstracting across device specific configuration files generating headers and links they reach and finally calculating a minimum set of test packets. It is not designed to identify failures caused from failed links and routers, bugs caused from faulty router hardware or software, and performance problems. The common causes of network failure are hardware failures and software bugs, in which that problems manifest both as reachability failures and throughput/latency degradation. To overcome this we are proposing new system.

IV. PROPOSED SYSTEM

Fig.2 shows the architecture of proposed system

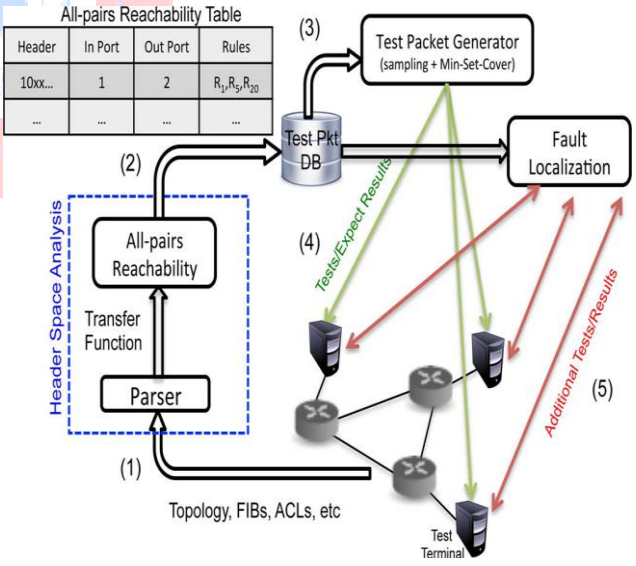

Fig. 2. ATPG system block diagram.

Based on the network model, 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. Fig. 5 is a block diagram of theATPG system. The system first collects

all the forwarding state from the network (step 1). This usually involves reading the FIBs, ACLs, and config files, as well as obtaining the topology. ATPG uses Header Space Analysis [16] to compute reachability between all the test terminals (step 2). The result is then used by the test packet selection algorithm to compute a minimal set of test packets that can test all rules (step 3). These packets will be sent periodically by the test terminals (step 4). If an error is detected, the fault localization algorithm is invoked to narrow down the cause of the error (step 5). While steps 1 and 2 are described in [16], steps 3–5 are new.

V. IMPLEMENTATION

A. Modules:

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

B. 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 Reachability 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 reachability 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. He
ader 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.

IV. RESULTS

After implementing the proposed system on java platform using Eclipse, the results obtained are as follows:

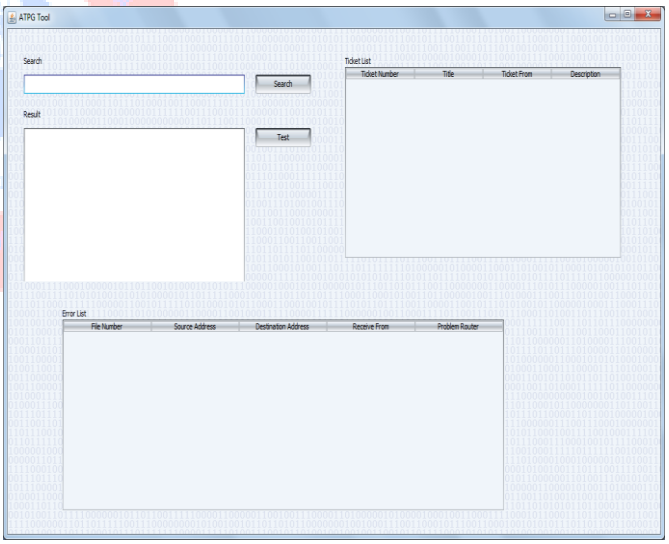

Fig.3. ATPG Tool Window

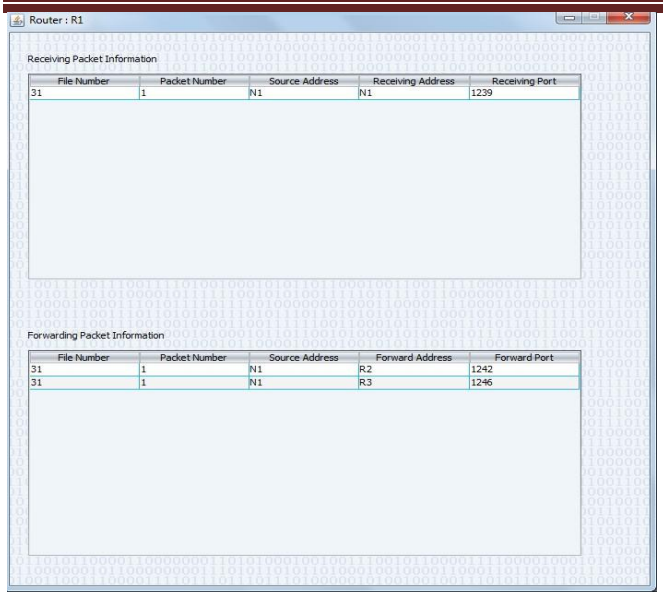

Fig. 4. Node Window

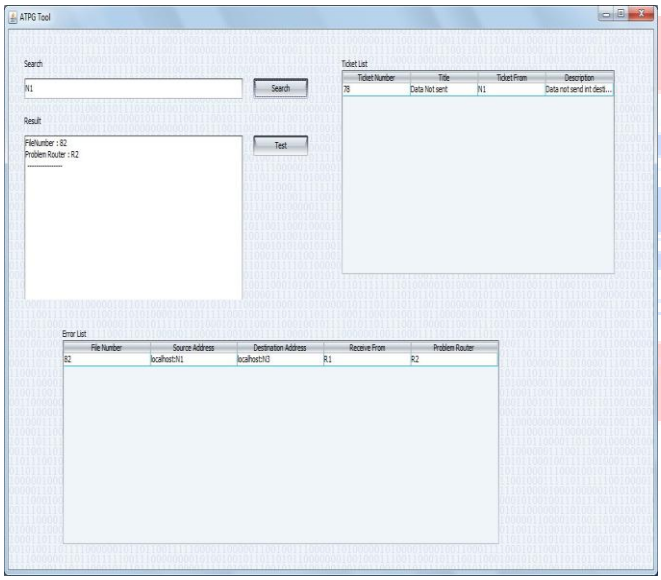

Fig. 5. ATPG Tool Localizing the Fault

After receiving a ticket from the node 1 this ATPG tool generates test packet and sends it to all the nodes in that path, based on the acknowledgement of all the nodes it locates the fault and displays the node where the problem has been found.

V. Conclusions

In current System it uses a method that is neither exhaustive nor scalable. Though it reaches all pairs of edge nodes it could not detect faults in liveness properties. ATPG goes much further than liveness testing with same framework. ATPG could test for reachability policy (by checking all rules including drop rules) and performance measure (by associating performance measures such as latency and loss of test packets). Our implementation also enlarges testing with simple fault localization scheme also build using header space framework.

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.
[7] M. Canini,D.Venzano, P. Peresini,D.Kostic, and J. Rexford, "A NICE way to test OpenFlow applications," in Proc. NSDI, 2012, pp. 10–10.
[8] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot, "Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data," in Proc. ACM CoNEXT, 2007, pp. 18:1–18:12..
[9] N. Duffield, "Network tomography of binary network performance characteristics," IEEE Trans. Inf. Theory, vol. 52, no. 12, pp. 5373–5388, Dec. 2006.
[10] N. Duffield, F. L. Presti, V. Paxson, and D. Towsley, "Inferring link loss using striped unicast probes," in Proc. IEEE INFOCOM, 2001, vol. 2, pp. 915–923.
[11] N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," IEEE/ACM Trans. Netw., vol. 9, no. 3, pp. 280–292, Jun. 2001.
[12] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in Proc. ACM SIGCOMM, 2011, pp. 350–361.
[13] "Hassel, the Header Space Library," [Online]. Available: https://bitbucket. org/peymank/hassel-public/
[14] Internet2, Ann Arbor, MI, USA, "The Internet2 observatory data collections," [Online]. Available: http://www.internet2.edu/observatory/ archive/data-collections.html
[15] M. Jain and 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.
[16] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in Proc. NSDI, 2012, pp. 9–9.
[17] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, "IP fault localization via risk modeling," in Proc. NSDI, Berkeley, CA, USA, 2005, vol. 2, pp. 57–70.
[18] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A SOFT way for OpenFlow switch interoperability testing," in Proc. ACM CoNEXT, 2012, pp. 265–276.
[19] K. Lai and M. Baker, "Nettimer: A tool for measuring bottleneck link, bandwidth," in Proc. USITS, Berkeley, CA, USA, 2001, vol. 3, pp. 11–11.
[20] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in Proc. Hotnets, 2010, pp. 19:1–19:6.