AUTOMATIC TEST PACKET GENERATION

M.P.Jegadeesan[1], Dr.G.Tholkappia Arasu[2]

[1]PG Scholar, Department of Computer Science and Engineering, AVS Engineering College
E-mail :jegadeesan.p@gmail.com

[2]Principal, AVS Engineering College
E-mail :tholsg@gmail.com

Abstract

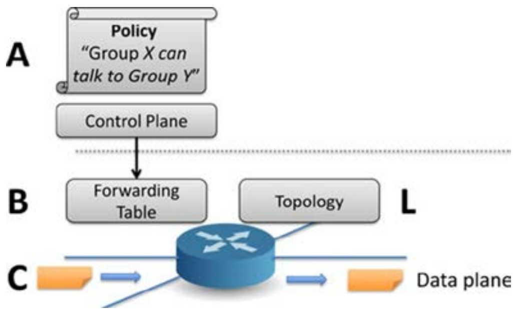
In this paper, We propose an automated and

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 (e.g., incorrect firewall rule) and performance problems (e.g., congested queue). ATPG complements but goes beyond earlier work in static checking (which cannot detect liveness or performance faults) or fault localization (which only localize faults given liveness results). We describe our prototype ATPG implementation and results on two real-world data sets: Stanford University's backbone network and Internet2. We find that a small number of test packets suffices to test all rules in these networks: For example, 4000 packets can cover all rules in Stanford backbone network, while 54 are enough to cover all links. Sending 4000 test packets 10 times per second consumes less than 1% of link capacity. ATPG code and the data sets are publicly available.

Index Terms—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 rulesets, the number of test packets needed is sur- prisingly small. For the Stanford network with over 757 000 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 liveness testing every millisecond using 1% of net- work bandwidth.

The contributions of this paper are as follows:

1) a survey of network operators revealing common failures and root causes(Section II);

2) a test packet generation algorithm (Section IV-A);

3) a fault localization algorithm to isolate faulty devices and rules (Section IV-B);

4) ATPG use cases for functional and performance testing (Section V);

5) evaluation of a prototype ATPG system using rulesets collected from the Stanford and Internet2 backbones (Sections VI and VII).

2. 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.

Disadvantages Of Existing System

Not designed to identify liveness failures, bugs router hardware or software, or performance problems.

The two most common causes of network failure are hardware failures and software bugs, and that problems manifest themselves both as reachability failures and throughput/latency degradation.

3. PROPOSED SYSTEM

Automatic Test Packet Generation (ATPG) framework that automatically generates a minimal set of packets to test the liveness 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 liveness.

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 SYSTEM ARCHITECHURE

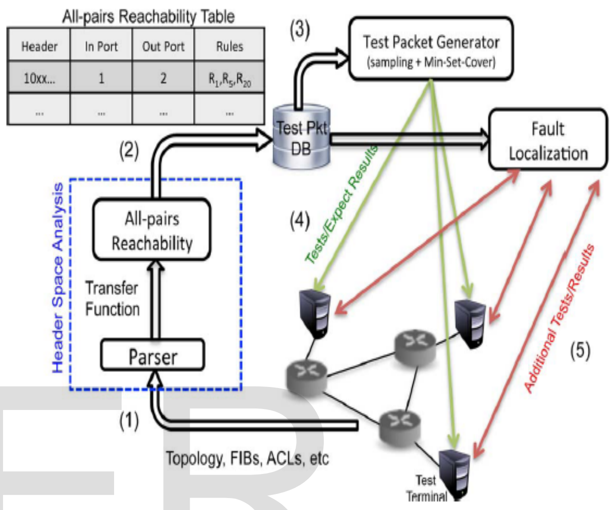


Fig.1. Automatic Test Packet Generation

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
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted

- Multithreaded
- Robust
- Dynamic

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.

IMPLEMENTATION

MODULES:

- Test Packet Generation
- Generate All-Pairs Reachability 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 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. 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.

5. CONCLUSION

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 [30]. 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 reachability), and finally determining a minimum set of test packets (Min-Set-Cover). Even the fundamental problem of automatically generating test packets for efficient liveness testing requires techniques akin to ATPG.

ATPG, however, goes much further than liveness testing with the same framework. ATPG can test for reachability policy (by testing all rules including drop rules) and performance health (by associating performance measures such as latency and loss with test packets). Our implementation also augments testing with a simple fault localization scheme also constructed using the header space framework. As in software testing, the formal model helps maximize test coverage while minimizing test packets. Our results show that all forwarding rules in Stanford backbone or Internet2 can be exercised by a surprisingly small number of test packets (for Stanford, and for Internet2).

Network managers today use primitive tools such as and. 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 awell-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.

.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.