International Journal of Research

International Journal of Research (IJR)

e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 11, November 2015

Available at http://internationaljournalofresearch.org

Automatic Test Packet Generation

Mr. K.L.Narsimha Rao [1]& A Chiranjeevi [2]

[1] Associate Professor, Dept of CSE, Marri Laxman Reddy Institute Of Technology & Management, Hyderabad, Telangana

[2] M.Tech CS (PG Scholar), Dept of CSE, Marri Laxman Reddy Institute Of Technology & Management, Hyderabad, Telangana

ABSTRACT

Networks are getting larger and more complex, yet administrators rely on rudimentary tools such as and to debug problems. 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 datasets are publicly available.

Index Terms—Data plane analysis; network troubleshooting; test packet generation

I. INTRODUCTION

Its 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 networks to misbehave or fail completely. Network engineers hunt down bugs using the most rudimentary tools and track down root causes using a combination of accrued wisdom and intuition. Debugging networks is only becoming harder as networks are getting *bigger* and are getting *more complicated* with over 6000 RFCs, router software is based on millions of lines of source code, and network chips often contain billions of gates It is a mall wonder that network engineers have been labeled "masters of complexity. The main contribution of this paper is what we call an 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. In Example 1, instead of Alice manually deciding which packets to send, the tool does so periodically on her behalf. In Example 2, the tool determines that itmust send packets with certain headers to "exercise" the video queue, and then determines that these packets are being dropped.

Example:

We tested our method on two real-world data sets—the backbone 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 surprisingly 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 network bandwidth 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. of the ATPG system. The system first collects all the forwarding state from the network this usually involves reading the FIBs, ACLs, and config files, as well as obtaining the topology. ATPG uses Header Space Analysis to compute reachability between all the test terminals The result is then used by the test packet selection algorithm to compute a minimal set of test packets that can test These packets will be sent periodically by the test terminals If an error is detected, the fault localization algorithm is invoked to narrow down the cause of the error described Networking is the word basically relating to computers and their connectivity. It is very often used in the world of computers and their use in different connections. The term networking implies the link between two or more computers and their devices, with the vital purpose of sharing the data stored in the computers, with each other. The networks between the computing devices are very common these days due to the launch of various hardware and computer software which aid in making the activity much more convenient to build and use.

General Network Techniques - When computers communicate on a network, they send out data packets without knowing if anyone is listening. Computers in a network all have a connection to the network and that is called to be connected to a network bus. What one computer sends out will reach all the other computers on the local network. For the different computers to be able to distinguish between each other, every computer has a unique ID called MAC-address (Media Access Control Address). This address is not only unique on your network but unique for all devices that can be hooked up to a network. The MAC-address is tied to the hardware and has nothing to do with IP-addresses. Since all computers on the network receives everything that is sent out from all other computers the MAC-addresses is primarily used by the computers to filter out incoming network traffic that is addressed to the individual computer.

When a computer communicates with another computer on the network, it sends out both the other computers MAC-address and the MAC-address of its own. In that way the receiving computer will not only recognize that this packet is for me but also, who sent this data packet so a return response can be sent to the sender.

On an Ethernet network as described here, all computers hear all network traffic since they are connected to the same bus. This network structure is called multi-drop.

One problem with this network structure is that when you have, let say ten (10) computers on a network and they communicate frequently and due to that they sends out there data packets randomly, collisions occur when two or more computers sends data at the same time. When that happens data gets corrupted and has to be resent. On a network that is heavy loaded even the resent packets collide with other packets and have to be resent again. In reality this soon becomes a bandwidth problem. If several computers communicate with each other at high speed they may not be able to utilize more than 25% of the total network bandwidth since the rest of the bandwidth is used for resending previously corrupted packets. The way to minimize this problem is to use network switches.

II. RELATED WORK

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.
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. We will only consider action faults because they cover most likely failure conditions and can be detected using only one test packet per rule. We leave match faults for future work.

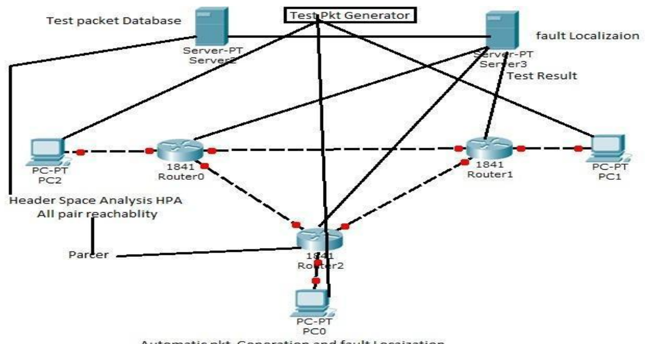
We can typically only observe a packet at the edge of the network after it has been processed by every matching rule we define an end-to-end version of the result function

The test packet generator, written in Python, contains a Cisco IOS configuration parser and a Juniper Junos parser. The data plane information, including router configurations, FIBs, MAC learning tables, and network topologies, is collected and parsed through the command line interface or XML files The generator then uses the Hassel header space analysis library to construct switch and topology functions We also found that 100% *link* coverage (instead of *rule* coverage) only needed 54 packets for Stanford The table also shows the large benefit gained by compressing the number of test packets—in most cases, the total number of test packets is reduced by a factor of 20–100 using the minimum set cover algorithm. This compression may make proactive link testing feasible for large networks

III. 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.
Based on the network model, ATPG generates the minimal number of test packets so that every forwarding rule in the network is check 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 [1].

Fig.1 is a block diagram of the ATPG system. The system first collects all the forwarding state from the network then all below test perform on network.



Automatic pkt Generation and fault Locaization

Step 1- This involves reading the FIBs, ACLs, and config file, and obtaining the topology. ATPG uses Header Space Analysis to compute reach ability 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 ll rules.

Step 3 - These packets will be sent periodically by the test terminals

Step 4 - If an error is erected, the fault localization algorithm is down the cause of the error.

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.

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 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.

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.

[21] F. Le, S. Lee, T. Wong, H. S. Kim, and D. Newcomb, "Detecting network-wide and router-specific misconfigurations through data mining," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 66–79, Feb. 2009.

[22] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iplane: An information plane for distributed services," in *Proc. OSDI*, Berkeley, CA, USA, 2006, pp. 367–380.

[23] A. Mahimkar, Z. Ge, J. Wang, J. Yates, Y. Zhang, J. Emmons, B.

Huntley, and M. Stockert, “Rapid detection of maintenance induced changes in service performance,” in *Proc. ACM CoNEXT*, 2011, pp. 13:1–13:12.

[24] A. Mahimkar, J. Yates, Y. Zhang, A. Shaikh, J.Wang, Z. Ge, and C. T. Ee, “Troubleshooting chronic conditions in large IP networks,” in *Proc. ACM CoNEXT*, 2008, pp. 2:1–2:12.

[25] H. Mai, A. Khurshid, R. Agarwal, M. Caesar, P. B. Godfrey, and S. T. King, “Debugging the data plane with Anteater,” *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 290–301, Aug. 2011.

[26] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, “Characterization of failures in an operational ip backbone network,” *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 749–762, Aug. 2008.

[27] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: Enabling innovation in campus networks,” *Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[28] “OnTimeMeasure,” [Online]. Available: http://ontime.oar.net/

[29] “Open vSwitch,” [Online]. Available: http://openvswitch.org/

[30] H. Weatherspoon, “All-pairs ping service for PlanetLab ceased,” 2005 [Online]. Available: http://lists.planet-lab.org/pipermail/users/2005-July/001518.html

[31] M.Reitblatt,N.Foster, J. Rexford, C. Schlesinger, andD.Walker, “Abstractions for network update,” in *Proc. ACM SIGCOMM*, 2012, pp. 323–334.

[32] S. Shenker, “The future of networking, and the past of protocols,” 2011 [Online].Available: http://opennetsummit.org/archives/oct11/shenkertue Pdf

[33] “Troubleshooting the network survey,” 2012 [Online]. Available: http://eastzone.github.com/atpg/docs/NetDebugSurvey.pdf

[34] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, “California fault lines: Understanding the causes and impact of network failures,” *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 315–326, Aug. 2010.

[35] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, and S.-J. Lee, “S3: A scalable sensing service for monitoring large networked systems,” in *Proc. INM*, 2006, pp. 71–76.