Automatic Test Packet Generation by using Fault Localization

Amruta S. Solanke[1], Sachin B. Jadhav[2], Vishakha V. Kharche[3]

[1]*Post Graduate Student, Department of CE, Padm. Dr. V.B.K.C.O.E., Malkapur, S.G.B.A. University, Maharashtra, India*

[2]*Asst. Professor, Department of CSE, Padm, Dr. V.B.K.C.O.E., Malkapur, S.G.B.A. University, Maharashtra, India*
[3]*Post Graduate Student, Department of CE, Padm. Dr. V.B.K.C.O.E., Malkapur, S.G.B.A. University, Maharashtra, India*

[1]amrutasolanke@gmail.com, [2]sachin.bjadhav@gmail.com, [3] vishakhakharche@gmail.com

Abstract— **In today's era, working on network is complex process as many users including naïve users, administrators etc. uses tools like ping and traceroute to debug problems. We have studieda systematic and automated approach for testing and debugging networks called An Automatic Test Packet Generation by using fault localization. ATPG interpret the router configurations and generates device-independent model. The model is used to generate a least possible set of test packets to exercise each link in the network or exercise each rule in the network. The Test packets are sent periodically and if detected failures trigger a distinct mechanism to localize faults. ATPG can detect mutually functional and performance problems. ATPG complements but it goes beyond previous work in static checking or fault localization. It is used for testing theliveness of the underlying topology and the congruence among data plane state and configuration specification. In this paper the small number of test packets suffices to test whole rules in these networks. ATPG code and datasets are available publicly.**

Keywords— Network troubleshooting, Test packet generation, Fault Localization.

I. INTRODUCTION

In networking the process of debugging is gettinga tough task. Every day, network engineers are wrestling with the router misconfigurations, fibercuts, software bugs, faulty interfaces, mislabelled cables, intermittent links, and several other reasons that are causing networks to misbehave or fail completely. Network engineers hunt down bugs using the most elementary tools such as ping, traceroute, SNMP and tcpdump track down root causes using a combination of accrued perception and intuition. The network debugging is becoming harder as networks are getting bigger (The modern data hubs may contain 10000 switches and a campus network may serve 50000 users, a 100-Gb/s long-haul and that link may transmit100000 flows) and are getting more complicated(with over 6000 RFCs, router software is based on the millions of lines of source code, and the network chips frequently contain billions of gates)[1]. For this consider an example.

Example 1: Suppose a router starts dropping packets silently with a faulty line card. An admin, who administers 100 routers, receives ticket from several unsatisfied users complaining about connectivity. Firstly Admin examines each router to see that, if the configuration was changed recently and concludes that the configuration was untouched [2].

Consequently, admin have to uses his knowledge of topology to trace the faulty device with ping and traceroute command tools. Finally, he have to call a colleague to replace the cable. Generally hardware failures and software bugs are the two most common causes of network failure, and that problems detected themselves both as reach ability failures and the throughput/latency degradation. Our goal is to detect these types of failures automatically. The main contribution of this paper is the Automatic Test Packet Generation [ATPG] framework that generates minimal set of packets to test the liveness of network automatically that provides support for topology and also automatically generate packets to test the performance affirmations such as packet latency.

In Example 1, admin manually decide which packets to send, which can does by the tool periodically on behalf of admin. ATPG detects and diagnoses errors by autonomously and testing all forwarding entries, firewalls rules, and several packet processing rules in network.

Network troubleshooting is difficult for following three reasons. First, the forwarding state which is distributed across over multiple routers and firewalls and gets defined by their forwarding tables, filter rules, and additional configuration parameters. Second, it is hard to observe forwardingstate because it requires logging into every box in the network manually. Third, the forwarding state is updating simultaneously, due to many different programs and protocols.

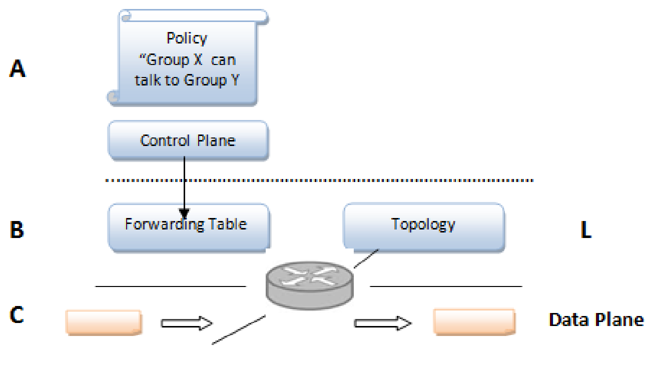

Figure. 1. Simplified view of Network.

Fig. 1 shows simplified view of network states. At the bottom of the figure the forwarding state used to forward each packet, containing L2 and L3 forwarding information base (FIB), access control lists, etc. The forwarding state which is written by the control plane can be local or remote as in the SDN model [3] and should properly implements the network administrator's policy. Policy examples include: "Security group X is inaccessible from security Group Y", "The Use of OSPF for routing," and "Video traffic must receive at least 1 Mb/s". We can consider the controller compiling the policy (A) into device specification config files (B), which in turn regulate the forwarding behaviour of every packet (C). To ensure the network performance as designed, whole three steps should remain stable at all times, i.e., A = B = C. In adding, the topology shown at the bottom right in the figure, as well satisfy a set of liveness properties L. Minimally, L requires that enough links and nodes that are working; if the control plane specifies that a laptop can access a server, the desired result can fail if linksfail. L can also specifies the performance guarantees that detect flaky links.

Recently, to check that A = B, researchers have proposed tools, enforces the consistency between policy and configuration[4], [5], [6], [7]. While these approaches can catch or prevent software logic errors in the control plane, they are not designed for identifying liveness failures caused by failed links and routers, bugs which caused by faulty router hardware/software/performance problems caused by network congestion. Such failures requires checking for L and whether B = C.

In ATPG, from the device configuration files and FIBs the test packets are generated algorithmically, for complete coverage required the minimum number of packets. Test packets are fed into the network so that each rule is getting directly exercised from the data plane. Since ATPG treats links just akin to normal forwarding rules, it guarantees testing of every link in the network due to full coverage. It also specialized to generate a minimal set of packets that simply test every link for checking liveness of network. At least in this basic form, we feel that ATPG or some similar techniques are fundamental to the network: Instead of reacting to failures, many network operators such as Internet2 [8] proactively check the strength of their network using pings between all pairs of sources. However, all-pairs ping does not guarantees testing of all links and has been found to be unscalable for large networks such as Planet Lab[9].

Organizations can customize ATPG to meet their needs; for example, they can choose to check only for network liveness(link cover) or check every rule (rule cover) to make certain security policy. ATPG can be customized to check merely for reachability or for performance. ATPG can adapt to constraints such as requiring test packets from only rare places in the network or using special routers to generate test packets from every port. ATPG can also be regulated to allocate more test packets to exercise more critical rules.

II. LITERATURE REVIEW

To understand the problems network engineer's encounters, and how they troubleshoot them at this time, we called subscribers to the NANOG1 mailing list for completing a survey in May–June 2012. Of the 61 who responded, 12 administer small networks (< 1 k hosts), 23 medium networks (1 k–10 k hosts), 11 large networks (10 k–100 k hosts), and 12 very large networks (> 100 k hosts). All responses are reported in [10] and are summarized in Table I. The most relevant findings are as follows.

TABLE I
Ranking Of Symptoms And Causes Reported By Administrators.

Category	Avg	% of ≥ 4
Reachability Failure	3.67	56.90 %
Throughput/Latency	3.39	52.54%
Intermittent Connectivity	3.38	53.45%
Router CPU High Utilization	2.87	31.67%
Congestion	2.65	28.07%
Security policy Violation	2.33	17.54%
Switch/Router Software Bug	3.12	40.35%
Hardware Failure	3.07	41.07%
Attack	2.67	29.82%
Software Upgrade	2.35	18.52%
Protocol Misconfiguration	2.29	23.64%
Q_oS/TE Misconfig	1.70	7.41%

Symptoms: From the most common six symptoms, from that four cannot be detected by static checks of the type A=B i.e. throughput/latency, router CPU utilization, intermittent connectivity, congestion and require ATPG-like dynamic testing. Even the enduring two failures reachability failure and security strategy violation might necessitate dynamic testing for detection of forwarding plane failures.

Causes: By vigorous checking there are two most common symptoms which are switch and router software bugs and hardware failure best found.

Cost of troubleshooting: The cost of network debugging the number of network-related tickets per month and the average time consumed to resolve a ticket by two metrics from [1].

TABLE II
Tools Used By Network Administrators

Category	Avg	% of ≥ 4
ping	4.50	86.67%
traceroute	4.18	80.00%
SNMP	3.83	60.10%
Configuration Version Control	2.96	37.50%
Netperf/iperf	2.35	17.31%
sFlow/NetFlow	2.60	26.92%

Tools: Table II shows that traceroute, ping, and SNMP are the most popular tools. When the question arises that what an ideal tool for network debugging would be, 70.7% reported a desire for automatic test generation for checking performance and correctness. Some added a aspiration for "long running tests to detect jitter or intermittent problems", "real-time link capacity observing", and "observing tools for network state."

In summary, our survey is small; it supports the hypothesis that network administrators have to face complex symptoms and causes. The debugging cost is important due to the frequency of problems and the time required solving these problems. Classical tools such as traceroute and ping are still heavily used, but an administrator desires more sophisticated tools.

III. PROPOSED SYSTEM

A. ATPG System:

Based on the network model, Minimum number of test packets are generated by ATPG so that every forwarding rule in the network is exercised and enclosed by at least one test packet. When an error is detected, ATPG uses a fault localization algorithm to conclude the links or failing rules.

Fig. 2 is a block diagram of the ATPG system. ATPG goes through the following steps: The system first collects all the forwarding state from the network (Step1). ATPG uses Header Space Analysis to calculate reachability between total test terminals (Step2). The result is then used by the test packet selection algorithm to calculate a minimum set of test packets that can test all rules (Step3). Test terminal send these packets periodically. If an error is identified, the fault localization algorithm is invoked to narrow down the cause of the error (Step 5).

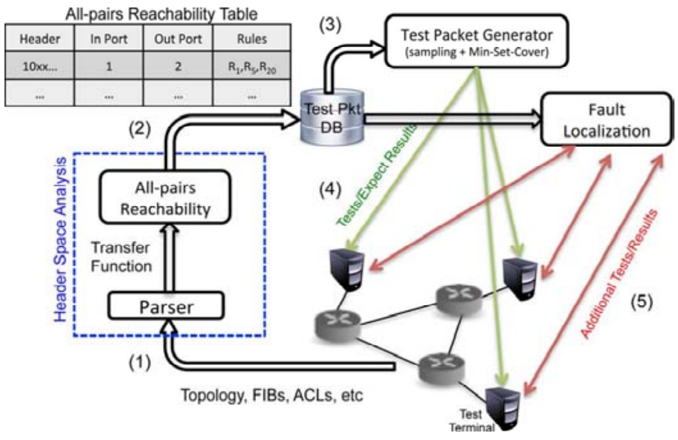

Fig. 2. ATPG system block diagram.

Step 1: Collect all forwarding states: Forwarding table which usually involves reading the FIBs (Forwarding Information States), ACLs (Access Control Lists), and config files, as well as obtaining the topology.

Step 2: Generate All-Pairs Reachability Table: ATPG Start's by computing the complete set of packet headers that can be sent from each test terminal to every other test terminal. For each and every such header, ATPG finds the complete set of rules it exercises along the path. To do so, all-pairs reachability algorithm applied by ATPG as follows:

1. Header constraints are applied. For example, if traffic can be sent on VLAN A, the instead of starting with an all- x header, the VLAN tag bits are set to A.
2. Protocol that match the packet are recorded in packet history. Hence all-pairs reachability table as shown in table III.

TableIII ALL-PAIRS REACHABILITY TABLE: ALL POSSIBLE HEADERS FROM EVERY TERMINAL TO EVERY OTHER TERMINAL, ALONG WITH THE RULES THEY EXERCISE

Header	Ingress Port	Egress Port	Rule History
h_1	P_{11}	P_{12}	$[r_{11}, r_{12}, \dots]$
h_2	p_{21}	p_{22}	$[r_{21}, r_{22}, \dots]$
…	…	…	…
h_n	p_{n1}	p_{n2}	$[r_{n1}, r_{n2}, \dots]$

Therefore total packets matching this class of header will come across the set of switch rules.

Step 3: Test Packet Generation: We send the set of test terminal in the network and test packets are receive. Our objective is to generate a set of test packets to employment every rule in every switch function, so that any fault will be observed by at least one test packet. This is similar to software test groups that try to test every possible branch in a program. The wider objective can be limited to testing every queue or every link.

When generating test packets, ATPG must contain two key Limitations:

1) *Port:* ATPG must only use test terminals that are available.

2) *Header: ATPG* essential only use headers that each test terminal is allowed to send.

Such as, the network administrator might only allow using a precise set of VLANs. Properly, we have the following difficult.

Problem (TPS): For a network with the switch functions $\{T_1, T_2..T_n\}$,and the topology function, Define the least set of test packets to exercise all nearby rules, subject to the port and header constraints. ATPG chooses test packets using an procedure we call Test Packet Selection). TPS first finds all corresponding classes among each pair of obtainable ports .A corresponding class is a set of packets that exercises the same combination of rules. It every class to select test packets, lastly compresses the resulting set of test packets to find the least covering set.

B. Fault Localization

ATPG sporadically sends a set of test packets. If those packets fail, ATPG pinpoints the mistake(s) that caused the problem.

1) Fault Model: A rule fails if its detected behavior varies from its expected behavior. ATPG retains track of where rules fail using a result function. For a rule, the outcome function is defined as

$$R(r,pk) = \begin{cases} 0, & if\ pk\ fails\ at\ rule\ r \\ 1, & if\ pk\ succeed\ at\ rule\ r \end{cases}$$

We divide faults into two categories: action faults and match faults. An action fault occurs when each packet identical to rule is processed inaccurately. Action faults include unpredicted packet defeat, a missing rule, congestion, and miswiring. On the alternative side, match mistakes are not easier to detect because they merely affect approximately packets matching the rule: such as example, when a rule matches a header it could not, or a rule misses a header it could match. We will only consider action faults because they cover most probable failure circumstances and can be noticed by using merely one test packet per rule.

2) Problem 2 (Fault Localization): A list of given $(pk_0, (pk_0),$ $(pk_1, (R(pk_1))$ … tuples, find all that satisfies $\exists pk_i,$ $R(pk_i, r) = 0$.

Step 1: Consider the outcomes from sending the regular test packets. For each passing test it place all rules they exercise into a set of passing rules, *P*. Similarly, for every failing test, place all rules they exercise into a set of potentially failing rules *F*. By our statement, one or more than one of the rules F are in error. So F-P, is a set of suspect rules.
Step 2: ATPG ensuing trims the set of suspect rules by weeding out properly working rules. ATPG does this using the reserved packets .ATPG selects reticent packets whose rule histories contain accurately one rule from the suspect set and sends these packets. Assume a reserved packet *p* exercises individual rule *r* in the suspect set. If the sending of *p* fails, ATPG infers that rule *r* is in error; if *p* passes; *r* is detached from the suspect set. ATPG repeats this process for every reserved packet selected in Step 2.
Step 3: In some possibilities, the suspect set is slight enough after Step 2, which ATPG can dismiss and report the suspect set. If desired, ATPG can narrow down the suspect set more by transferring test packets that exercise two or more than that of the rules in the suspect set by means of the same technique underlying Step 2. If those test packets pass, ATPG concludes that none of the exercised rules are in error and eliminates those rules from the suspect set.
If our Fault Propagation assumption clutches, the technique shall not miss any faults, and so it will have no *false negatives*.
False Positives: Note that the localization technique may present *false positives*, rules left in the suspect set at the end of Step 3. Exactly, one or more than one rules in the suspect set may in fact behave properly. False positives are inevitable in some possibilities.

When two rules are in sequence and there is no path to exercise only one of them, we can say the rules are not distinguishable any packet that exercises one rule shall also exercise the other. Hence forth, if only one rule fails, we can't tell which one. Such as if an ACL rule is monitored immediately by a forwarding rule that matches the same header, the two rules are not distinguishable. Notice that if we test terminals before and after each rule with sufficient test packets, we can differentiate each rule. Thus, the deployment of test terminals moves test coverage as well as localization accuracy.

IV. CONCLUSION

The network administrators having a fundamental problem to test the liveness of a network. They uses the basic tools such as traceroute and ping. To resolve the problem of automatically generated test packets for efficient liveness testing requires techniques an alogous to ATPG. The reachability policy and performance health can be tested by ATPG. By fault localization testing ATPG is getting augmented also constructed using the header space framework. So we hope that network ATPG will be evenly useful for automated dynamic testing of production networks.

REFERENCES

[1] Zeng , Kazemian, Varghese,and Nick "Automatic Test Packet Generation",VOL. 22, NO. 2, APRIL 2014.

[2] 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.

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

[4] 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.

[5] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis:Static checking for networks," in *Proc. NSDI*, 2012, pp. 9–9.

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

[7] M.Reitblatt,N.Foster, J. Rexford, C. Schlesinger, and D.Walker, "Abstractions for network update," in *Proc. ACM SIGCOMM*, 2012, pp. 323–334.

[8] Internet2, Ann Arbor, MI, USA, "The Internet2 observatory data collections,"[Online]. Available: http://www.internet2.edu/observatory/archive/data-collections.html

[9] H. Weather spoon, "All-pairs ping service for PlanetLab ceased," 2005[Online]. Available: http://lists.planet-lab.org/pipermail/users/2005-July/001518.html.

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