Quality Analysis in Data Transfer without Packet Dump

D.R. ManoRanjani[1] M.S. NishaPriya[2] BheemaMehraj[3]

[1]Dept. of C.S.E., Bharath University, Chennai, Tamil Nadu, INDIA.

[2]Dept. of C.S.E., Bharath University, Chennai, Tamil Nadu, INDIA.

[3]Assitant Professor, Dept. of C.S.E., Bharath University, Chennai – 600 073. INDIA.

ABSTRACT

"Programmed check Packet Generation" (ATPG). ATPG introduce switch configurations and creates a widget autonomous model. The model is employed to supply a base arrangement of check parcels to (insignificantly) follow every association within the system or (maximally) follow every church doctrine within the system. Check bundles area unit sent often and recognized disappointments trigger a distinct system to confine the disadvantage. ATPG will distinguish each utilitarian (e.g., wrong firewall tenet) and execution problems (e.g., engorged line). ATPG supplements nonetheless goes past previous add static checking (which cannot determine animateness or execution blames) or flaw limitation (which simply restrict flaws given animate ness results). We tend to portray our model ATPG execution and results on 2 true data sets: Stanford University's spine system and Internet2. We tend to find that to a small degree range of check parcels suffices to check all tips in these systems: for example 4000 bundles will cowl all standards in Stanford spine system whereas fifty four is adequate to hide all connections. Causation 4000 check bundles ten times every second expends but one hundred and twenty fifth of association limit. ATPG code and therefore the data sets area unit brazenly obtainable. The current framework offers a programmed checking arrangement of component host setup convention server for activity a top quality certification test to the element host style convention server. Keywords: Test Packet Generation, information Plane Analysis, Network Troubleshooting

1.INTRODUCTION

It is notoriously laborious to correct networks. each day net-work engineers wrestle with router misson durations, per cuts, faulty interfaces, mislabeled cables, package bugs, intermittent links and a myriad different reasons that cause networks to move, or fail fully. Network engineers search out bugs mistreatment the foremost rudimentary tools (e.g. ping, traceroute, SNMP, and tcpdump), and capture root causes employing a combination of accumulated knowledge and in-tuition. Debugging networks is barely turning into more durable as networks have gotten larger (modern knowledge centers might contain ten,000 switches, a field network might serve fifty,000 users, a one hundredGb/s long-term link might carry 100,000 owns), and have gotten a lot of sophisticated (with over half-dozen,000 RFCs, router package relies on innumerable lines of ASCII text file, and network chips typically contain billions of gates). Tiny marvel that network engineers are labelled masters of complexity".

Troubleshooting a network is tough for 3 reasons. First, the forwarding state is distributed across multiple routers and rewalls and by their forwarding tables, alter rules and different configuration parameters. Second, the forwarding state is difficult to look at, as a result of it generally needs manually work into each confine the network. Third, there are many various programs, protocols and humans change the forwarding state at the same time. Once Alice uses ping and traceroute, she is employing a crude lens to look at this forwarding state for clues to trace down the failure.Figure one could be asimplify read of network state. At the bottom of the configure is that the forwarding state wont to forward every packet, consisting of the L2 and L3 forwarding info base (FIB), access management lists, etc. The forwarding state is written by the management plane (that will be native or remote as within the SDN model), and may properly implement the network administrator's policy. samples of the policy include: Security cluster X is isolated from security cluster Y", Use OSPF for routing", and Video trace ought to receive a minimum of 1Mb/s".

The main contribution of this paper is what we tend to decision Associate in Nursing Automatic take a look at Packet Generation (ATPG) framework that mechanically generates a tokens set of packets to check the aliveness of the underlying topology and also the harmoniousness between knowledge plane state and configuration specifications. The tool also can mechanically generate packets to check performance assertions like packet latencies.

2. NETWORK MODEL

ATPG uses the header house framework | a geometrical model of however packets square measure processed we tend to delineated. In header house, protocol-specific meanings related to headers square measure ignored: a header is viewed as at sequence of ones and zeros. A header may be a purpose (and own may be a region) within the f0; 1gL house, wherever L is Associate in Nursing bound on header length. By mistreatment the header house

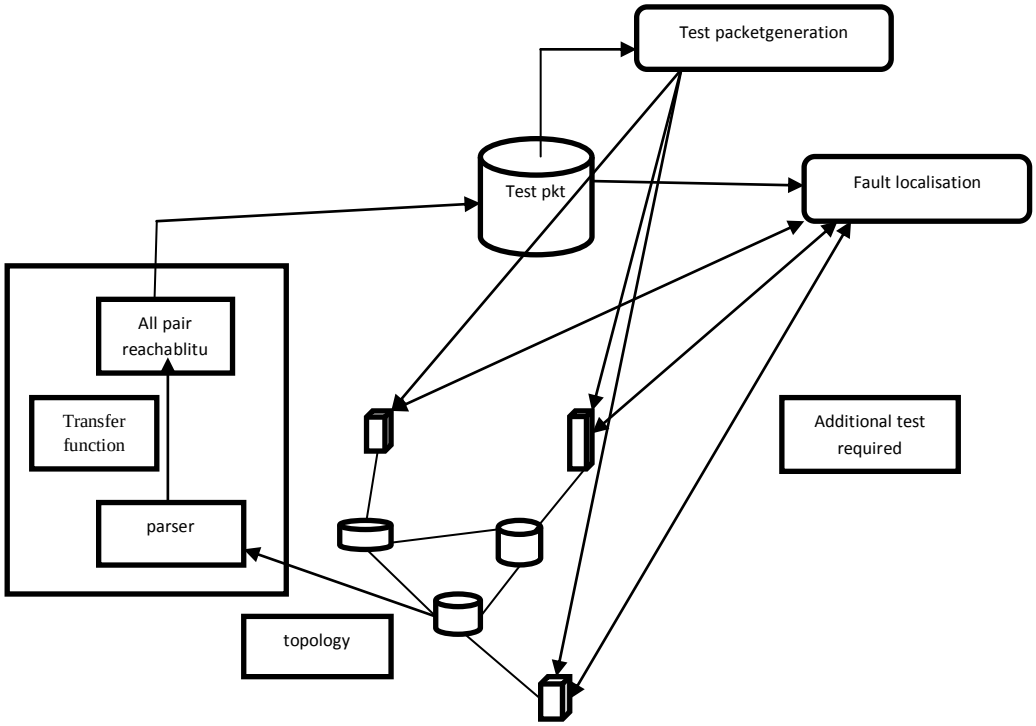
ISSN: 2278 – 1323

Framework, we tend to get a male function, vendor-independent and protocol-agnostic model of the network3 that simplify the packet generation method significantly.

Life of a packet

The lifetime of a packet will be viewed as applying the switch and topology transfer functions repeatedly. Once a packet pk arrives at a network port p, the switch operate T that contains the input port pk:p is applied to pk, producing a listing of latest packets [pk1; pk2; : : :]. If the packet reaches its destination, it's recorded. Otherwise, the topology function is employed to invoke the switch operate containing the new port. The method repeats till packets reach their destinations

3.ATPG SYSTEM



ATPG System block diagram

Based on the network model, ATPG consistently generates the token range of take a look at packets supported network state, so each forwarding rule the network is exercised and lined by a minimum of one take a look at packet. Once a mistake is detected, ATPG uses a fault localization algorithmic rule to and the failing rules or links. When generating take a look at packets, there are 2 main constraints:

(1) Port: ATPG should use solely take a look at terminals that are available;

(2) Header: ATPG should use solely headers that every take a look at terminal is permissible to send. As an example, the network administrator might solely enable employing a specific set of VLANs.

Step 1: Generate Associate in nursing all-pairs reachability table.

Step 2: Sampling.

Step 3: Compression.

Properties

Proof Sketch

Property a pair of (Completeness).

Property three (Polynomial Runtime).

Proof Sketch.

Fault Localization

ATPG picks and sporadically sends a group of take a look at packets. If take a look at packets fail we tend to conjointly ought to pinpoint the fault(s) that caused the matter.

Fault model

A rule fails if its determined behavior is different from what we tend to expected. We tend to keep track of wherever rules fail employing a result operate R. For a selected rule, r, the result operate is American state as we tend to solve this drawback opportunistically and in steps.

ISSN: 2278 – 1323

Step 1: think about the results from our regular take a look at packets. For each passing take a look at, place all the foundations they exercise into the set of passing rules, P. If we tend to equally American state ne all rules traversed by failing take a look at packets F, then one or additional of the foundations in F are in error. So F P may be a set of suspect rules.
Step 2: we would like to form the set of suspect rules as little as attainable by hunting down all the properly operating rules. For this we tend to create use of the reserved packets.
Step 3: In most cases we've alittle enough suspect set that we will stop here and report all of them. If our Fault Propagation assumption holds, the strategy won't miss any faults, and so can don't have any false negatives.
False positives: The localization technique might introduce false positives, that ar left within the suspect set at the tip of Step three. Specifically, one or additional rules within the suspect set might actually behave properly.

4. FUNCTIONAL TESTING:

We can check the useful correctness of a network by testing that each approachable forwarding and drop rule out the network is behaving correctly:

Forward rule: A forwarding rule is behaving properly if a check packet exercises the rule, and leaves on the right port with the right header.
Link rule: A link rule could be a special case of a forwarding rule. It may be checked by ensuring a test packet passes properly over the link while not header modifications.
Drop rule: Testing drop rules is tougher as a result of we have a tendency to the absence of received check packets. We want to understand that check packets would possibly reach associate degree egress check terminal if a drop rule was to fail.

Performance testing

Congestion: If a queue is engorged, packets can experience longer queueing delays.
Bandwidth: equally, we will live the accessible information measure of a link, or for a specific service class. ATPG can generate the check packet headers we want to check each link, or each queue, or each service class; we have a tendency to then send a stream of packets with these headers to live the information measure.
Strict priorities: Likewise, we will confirm if 2 queues, or service categories, are in different strict priority categories. If they're, then packets sent within the lower priority category ought to ne'er a shock therapy the accessible information measure or latency of packets within the higher priority category.

5. IMPLEMENTATION

We implemented a prototype system to automatically parse router configurations and generate a set of test packets for the network.

Test packet generator

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 (Cisco IOS) or XML les (Junos). The generator then uses as header space analysis library to construct switch and topology functions.

Network monitor

The network monitor assumes there are special test agents in the network that are able to send/receive test packets.

6.EXPERIMENTS:

Data Sets: Stanford and Internet2

We evaluated our prototype system on two sets of network configurations: the Stanford University backbone and the Internet2 backbone, representing a mid-size enterprise network and a nationwide backbone network respectively.

Stanford Backbone: With a population of over 15,000 students, 2,000 faculty, and 16 IPv4 subnets, Stanford represents a large enterprise network. There are 14 operational zone (OZ) Cisco routers connected via 10 Ethernet switches to 2 backbone Cisco routers that in turn connect Stanford to the outside world. Overall, the network has more than 757,000 forwarding entries and 1,500 ACL rules.

Internet2: Internet2 is a nationwide backbone network with 9 Juniper T1600 routers and 100 Gb/s interfaces, supporting over 66,000 institutions in United States. There are about 100,000 IPv4 forwarding rules. All Internet2 configurations and FIBs of the core routers are publicly avail-able [10], with the exception of ACL rules, which are re-moved for security concerns. Although IPv6 and MPLS entries are also available, we only use IPv4 rules in this paper.

Test Packet Generation

We ran our ATPG tool on a quad core Intel Core i7 CPU 3.2GHz and 6GB memory, using 8 threads. For a given number of test terminals, we generated the minimum set of test packets needed to test all the reachable rules in the Stanford and Internet2 backbones. Table 5 shows the number of test packets we need to send.

ISSN: 2278 – 1323

Testing in Emulated Network

To evaluate the network monitor and test agents, we replicated the Stanford backbone network in Mininet, a container based network emulator. We used Open vSwitch (OVS) to emulate the routers, using the real port configuration information, and connected them according to the real topology. We then translated the forwarding entries in the Stanford backbone network into equivalent Open-Flow rules and installed them in the OVS switches with Beacon. We used emulated hosts to send and receive test packets generated by ATPG.

7. RESULTS

Congestion: We detect congestion by measuring the one-way latency of test packets. In our emulation environment, all terminals are synchronized to the host's clock so the latency can be calculated with a single time-stamp and one-way communication.To create congestion, we rate-limited all the links in the emulated Stanford network to 30Mb/s, and create two 20Mb/s UDP owns: poza to yoza at t = 0 and roza to yoza at t = 30s, which will congest the link bbra yoza starting at t = 30s. The queue inside the routers will build up and test packets will experience longer queuing delay. The bottom right graph next to pozb shows the latency experienced by two test packets, one from pozb to roza and the other one from pozb to yoza. At t = 30s, the bozb yoza test packet experience a much higher latency, correctly signaling congestion. Since these two test packets share the bozb s_1 and $s_1$bbra links, we can conclude that the congestion is not happening in these two links, therefore we can correctly infer that bbra yoza is the congested link. Available Bandwidth: ATPG can also be used to mon-itor available bandwidth. For this experiment, we used Path load, a bandwidth probing tool based on packet pairs/packet trains. We repeated the previous experiment, but decreased the two UDP owns to 10Mb/s, so that the bottleneck available bandwidth was 10Mb/s. Path load reports that bozb yoza has an available bandwidth of 11.715Mb/s, bozb roza has an available bandwidth of 19.935Mb/s, while the other (idle) terminals report 30.60Mb/s. Using the same argument as before, we automatically conclude that bbra yoza link is the bottleneck link with 10Mb/s available bandwidth.

Priority: We created priority queues in OVS using Linux's htb scheduler and tc utilities.

8. RELATED WORK

We are unaware of earlier techniques that automatically generate test packets from configurations. The closest related work we know of are online tools that check invariants of different components in networks. On the control plane, NICE attempts to exhaustively cover the code paths symbolically in controller applications with the help of simplifiedswitch/host models. On the data plane, Anteater models invariants as Boolean satisfy ability problems and checks them against configurations with a SAT solver. Header Space Analysis uses a geometric model to check reachability, detect loops, and verify slicing. Recently, SOFT is proposed to verify the consistency between different Open-Flow agent implementations that are responsible for bridging control and data planes in the SDN context. ATPG complements these checkers by directly testing the data plane and covering a significant set of dynamic or performance errors that cannot be otherwise captured.

End-to-end probes have long been used in network diagnosis. Du field uses Binary Tomography to detect the smallest set of failed links that explains end-to-end measurements. Net diagnoser further combines end-to-end probes with routing data. Researchers also use various models to correlate network metrics with network events. Recently, mining low-quality, unstructured data, such as router configurations and network tickets has attracted interest. By contrast, the primary contribution of ATPG is not fault localization, but determining a compact set of end-to-end measurements that can cover every link or every rule. Further, ATPG is not limited to liveness testing but can be applied to checking higher level properties such as performance.

Many approaches to develop a measurement-friendly architecture are proposed for managing large networks. It is also suggested that routers should be able to sample the packets for measurement. Our approach is complementary to these proposals: ATPG does not dictate the locations of injecting network probes and how these probes should be constructed. By incorporating input and port constraints, ATPG can generate test packets and injection points using existing deployment of measurement devices.

9. 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. It success to and a minimal set of end-to-end packets that traverse each link. Doing this requires a way of abstracting across device specific configuration les (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 goes much further than liveness testing, however, with the same framework. ATPG can test for reachability policy and performance health. 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 with minimum 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 (<4,000 for Stanford, and <40,000 for Internet2).

ISSN: 2278 – 1323

Network managers today use primitive tools such as ping and traceroute. Our survey results indicate that they are eager for more sophisticated tools. Other fields of engineering indicate that these demands 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 rules) and dynamic (e.g., timing) verification. We hope ATPG is a useful starting point for automated dynamic testing of production net-works.

10. REFERENCES

[1] ATPG code repository. http://eastzone.github.com/atpg/.

[2] Beacon. http://www.beaconcontroller.net/.

[3] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford. A NICE way to test open ow applications. Proceedings of the 9th conference on Symposium on Networked Systems Design & Implementation, 2012.

[4] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot. Netdiagnoser: troubleshooting network unreachabilities using end-to-end probes and routing data. In Proceedings of the 2007 ACM CoNEXT conference, CoNEXT '07, pages 18:1{18:12, New York, NY, USA, 2007. ACM.

[5] N. Du eld. Network tomography of binary network performance characteristics. Information Theory, IEEE Transactions on, 52(12):5373 {5388, dec. 2006.

[6] N. Du eld, F. Lo Presti, V. Paxson, and D. Towsley. Inferring link loss using striped unicast probes. In INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, volume 2, pages 915 {923 vol.2, 2001.

[7] N. G. Du eld and M. Grossglauser. Trajectory sampling for direct tra c observation. IEEE/ACM Trans. Netw., 9(3):280{292, June 2001.

[8] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In Proceedings of the ACM SIGCOMM 2011 conference, SIGCOMM '11, pages 350{361, New York, NY, USA, 2011. ACM.

[9] Hassel, the header space library. https://bitbucket.org/peymank/hassel-public/.

[10] The Internet2 Observatory Data Collections. http://www.internet2.edu/observatory/archive/ data-collections.html.

[11] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. IEEE/ACM Trans. Netw., 11(4):537{549, Aug. 2003.

[12] P. Kazemian, G. Varghese, and N. McKeown. Header Space Analysis: static checking for networks. Proceedings of the 9th conference on Symposium on Networked Systems Design & Implementation, 2012.

[13] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren. Ip fault localization via risk modeling. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2, NSDI'05, pages 57{70, Berkeley, CA, USA, 2005. USENIX Association.

[14] M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic. A SOFT way for open ow switch interoperability testing. In Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies, CoNEXT '12, 2012.

ISSN: 2278 – 1323