ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

A Noval Approach for ATPG Router Configuration

CH.Sireesha
M.Tech,
Center for Electronic Research & Development, Dept of Electronics & Communication Engineering, Mallareddy Institute of Engineering and Technology, Secunderabad, India.

R.Raja Kishore
Assistant Professor,
Center for Electronic Research & Development, Dept of Electronics & Communication Engineering, Mallareddy Institute of Engineering and Technology, Secunderabad, India.

Dr.M Narsing Yadav
Professor & HOD,
Center for Electronic Research & Development, Dept of Electronics & Communication Engineering, Mallareddy Institute of Engineering and Technology, Secunderabad, India.

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 find that a small number of test packets suffice 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 consume less than 1% of link capacity. ATPG code and the datasets are publicly available.

Index Terms:
Data plane analysis, network troubleshooting, test packet generation**.**

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 networks to misbehave or fail completely. Network engineers hunt down bugs using the most rudimentary tools (e.g., SNMP) and track down root causes using a combination of accrued wisdom and intuition. Debugging networks is only becoming harder as networks are getting bigger (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 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). 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. ATPG detects and diagnoses errors by independently and exhaustively testing all forwarding entries, firewall rules, and any packet processing rules in the network. In ATPG, test packets are generated algorithmically from the device configuration files and FIBs, with the minimum number of packets required for complete coverage.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Test packets are fed into the network so that every rule is exercised directly from the data plane. Since ATPG treats links just like normal forwarding rules, its full coverage guarantees testing of every link in the network. It can also be specialized to generate a minimal set of packets that merely test every link for network liveness. At least in this basic form, we feel that ATPG or some similar technique is fundamental to networks: Instead of reacting to failures, many network operators such as Internet2 [14] proactively check the health of their network using pings between all pairs of sources.

EXISTING SYSTEM

1. Testing liveness of a network is a fundamental problem for ISPs and large data centre 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).
2. To check enforcing consistency between policy and the configuration.

DISADVANTAGES OF EXISTING SYSTEM:

1. Not designed to identify liveness failures, bugs router hardware or software, or performance problems.
2. 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.

Definitions:

Below Fig. summarizes the definitions in our model.

Bit	$b = 0\|1\|\mathtt{x}$
Header	$h = [b_0, b_1, \ldots, b_L]$
Port	$p = 1\|2\|\ldots\|N\|\mathtt{drop}$
Packet	$pk = (p, h)$
Rule	$r : pk \rightarrow pk$ or $[pk]$
Match	$r.matchset : [pk]$
Transfer Function	$T : pk \rightarrow pk$ or $[pk]$
Topo Function	$\Gamma : (p_{src}, h) \rightarrow (p_{dst}, h)$

Packets:

A packet is defined by a (port, header) tuple, where the (port) denotes a packet's position in the network at any time instant; each physical port in the network is assigned a unique number.

Switches:

A switch transfer function, , models a network device, such as a switch or router. Each network device contains a set of forwarding rules (e.g., the forwarding table) that determine how packets are processed. An arriving packet is associated with exactly one rule by matching it against each rule in descending order of priority, and is dropped if no rule matches.

Rules:

A rule generates a list of one or more output packets, corresponding to the output port(s) to which the packet is sent, and defines how packet fields are modified. The rule abstraction models all real-world rules we know including IP forwarding (modifies port, checksum, and TTL, but not IP address); VLAN tagging (adds VLAN IDs to the header); and ACLs (block a header, or map to a queue). Essentially, a rule defines how a region of header space at the ingress (the set of packets matching the rule) is transformed into regions of header space at the egress [16].

Rule History:

At any point, each packet has a rule history: an ordered list of rules the packet matched so far as it traversed the network.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Rule histories are fundamental to ATPG, as they provide the basic raw material from which ATPG constructs tests.

Topology:

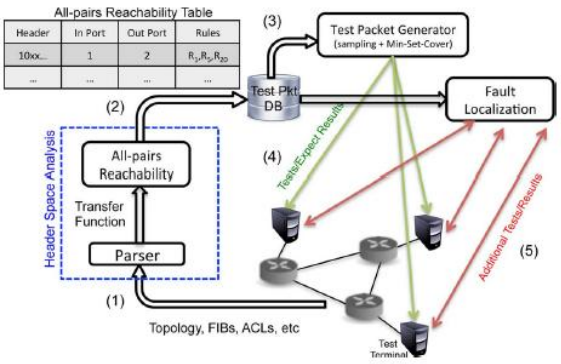
The topology transfer function, T, models the network topology by specifying which pairs of ports are connected by links. Links are rules that forward packets from Psrc to Pdst without modification. If no topology rules match an input port, the port is an edge port, and the packet has reached its destination.

Life of a Packet:

The life of a packet can be viewed as applying the switchand topology transfer functions repeatedly. When a packet arrives at a network port, the switch function that contains the input port is applied to, producing a list of new packets. If the packet reaches its destination, it is recorded. Otherwise, the topology function is used to invoke the switch function containing the new port. The process repeats until packets reach their destinations (or are dropped).

ATPG SYSTEM:

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.



Above is a block diagram of the ATPG system. The system first collects all the forwarding state from the network (step 1).

This usually involves reading the FIBs, ACLs, and configuration 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.

Test Packet Generation

1) Algorithm:

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

2) Header:

ATPG must only use headers that each test terminal is permitted to send. For example, the network administrator may only allow using a specific set of VLANs. Formally, we have the following problem.

Problem 1 (Test Packet Selection):

For a network with the switch functions, , and topology function, , determine the minimum set of test packets to exercise all reachable rules, subject to the port and header constraints. ATPG chooses test packets using an algorithm we call TestPacket Selection (TPS). TPS first finds all equivalent classes between each pair of available ports. An equivalent class is a set of packets that exercises the same combination of rules.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

It then samples each class to choose test packets, and finally compresses the resulting set of test packets to find the minimum coveringset.

2) Properties:

The TPS algorithm has the following useful properties.

Property 1 (Coverage):

The set of test packets exercise all reachable rules and respect all port and header constraints.

Proof Sketch:

Define a rule to be reachable if it can be exercised by at least one packet satisfying the header constraint, and can be received by at least one test terminal. A reachable rule must be in the all-pairs reachability table; thus, set cover will pick at least one packet that exercises this rule. Some rules are not reachable: For example, an IP prefix may be made unreachable by a set of more specific prefixes either deliberately (to provide backup) or accidentally (due to misconfiguration).

Property 2 (Near-Optimality):

The set of test packets selected by TPS is optimal within logarithmic factors among all tests giving complete coverage.

Proof Sketch:

This follows from the logarithmic (in the size of the set) approximation factor inherent in Greedy Set Cover.

Property 3 (Polynomial Runtime):

The complexity of finding test packets is where the number of test terminals is, is the network diameter, and is the average number of rules in each switch.

Proof Sketch: The complexity of computing reachability from one input port is, and this computation is repeated for each test terminal.

B. Fault Localization

ATPG periodically sends a set of test packets. If test packets fail, ATPG pinpoints the fault(s) that caused the problem.

***1)* Fault Model:**

A rule fails if its observed behavior differs from its expected behavior. ATPG keeps track of where rules fail using a result function. For a rule, the result function is defined as

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

"Success" and "failure" depend on the nature of the rule: Aforwarding rule fails if a test packet is not delivered to the intendedoutput 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.

2. Algorithm:

Our algorithm for pinpointing faulty rules assumesthat a test packet will succeed only if it succeeds at every hop. For intuition, a ping succeeds only when all the forwarding rules along the path behave correctly. Similarly, if a queue is congested, any packets that travel through it will incur higher latency and may fail an end-to-end test.

We solve this problem opportunistically and in steps.

Step 1: Consider the results from sending the regular testpackets. For every passing test, place all rules they exercise into a set of passing rules. Similarly, for every failing test, placeall rules they exercise into a set of potentially failing rules .By our assumption, one or more of the rules in F are in error. Therefore, F-P is a set of suspect rules.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

Step 2:ATPG next trims the set of suspect rules by weedingout correctly working rules. ATPG does this using the reservedpackets (the packets eliminated by Min-Set-Cover). ATPG selects reserved packets whose rule histories contain exactly onerule from the suspect set and sends these packets. Suppose a reserved packet exercises only rule in the suspect set. If the sending of fails, ATPG infers that rule is in error;

Step 3: In most cases, the suspect set is small enough afterStep 2, that ATPG can terminate and report the suspect set.If needed, ATPG can narrow down the suspect set further by sending test packets that exercise two or more of the rules in the suspect set using the same technique underlying Step 2. If these test packets pass, ATPG infers that none of the exercised rules are in error and removes these rules from the suspect set. If our Fault Propagation assumption holds, the method will not miss any faults, and therefore will have no false negatives. False Positives: Note that the localization method may introduce

False positives, rules left in the suspect set at the end of Step 3. Specifically, one or more rules in the suspect set may in fact behave correctly. False positives are unavoidable in some cases. When two rules are in series and there is no path to exercise only one of them, we say the rules are indistinguishable; any packet that exercises one rule will also exercise the other. Hence, if only one rule fails, we cannot tell which one. For example, if an ACL rule is followed immediately by a forwarding rule that matches the same header, the two rules are indistinguishable. Observe that if we have test terminals before and after each rule (impractical in many cases), with sufficient test packets, we can distinguish every rule. Thus, the deployment of test terminals not only affects test coverage, but also localization accuracy.

Proposed System:

Contender framework generates minimum no of packets automatically to debug the false occurring in the network model This tool could automatically generate packets for checking performance assertions such as like packet loss finds and determines errors by independently testing all forwarding entries any packet processing rules and security models in network test packets are generated algorithmically from device configuration files and from FIBs which requires minimum number of packets for complete coverage Test packets are fed into the network in which that every rule is covered directly from the data plane Since treats links like normal forwarding conditions its full coverage provides testing of every link in the network model It can also best specialized to form a minimal set of packets that obviously test every link for network likeness At least in this basic form, we would feel that some different technique is fundamental to networks Instead of reacting to failures many network operators such as proactivelycheck the health of their network using pings between all pairs of sources all-pairs does not provide testing of all links and has been found to be unsalable for large networks such as Planet Lab.

IV. Methodology The proposed system can be divided into following modules:

1. Failures and root causes of network operators
2. Data plane analysis
3. Network troubleshooting
4. ATPG system
5. Network Monitor

1. Failure and Root Causes of Network Operators:

Network traffic is represented to a specific queue in router but these packets are drizzled because the rate of token bucket low It is difficult to troubleshoot a network for three different models First the forwarding state is shared to multiple routers and security and is determined by the forwarding data filter conditions and configuration parameters Second the forwarding state is difficult to watch because it requires manually logging into every box in the network model Third the forwarding state is edited simultaneously by different programs protocols and humans.

2. Data Plane Analysis:

Automatic Test Packet Generation framework which automatically generates a minimum set of packets to check the likeness of underlying network models and

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

congruence different data plane state and configuration specifications These model can automatically generate packets to test performance assertions like packet latency ATPG find faults by independently and exhaustively checking all security rules forwarding entries and packet processing conditions in network. The test packets are generated algorithmically from the device configuration different files and FIBs, with less number of packets needed for whole coverage Test packets are fed in the network so that every rule is covered directly from the data plane. This tool can be customized to check only for reach ability or for its performance.

3. Network Troubleshooting:

The cost of network debugging is captured by two metrics. One is the number of network-related tickets per month and another is the average time taken to resolve a ticket. There are 35% of networks which generate more than 100 tickets per month. Of the respondents, 40.4% estimate takes under 30 minutes to resolve a ticket. If asked what is the ideal tool for network debugging it would be, 70.7% reports automatic test generation to check performance and correctness. Some of them added a desire for long running tests to find jitter or intermittent real-time link capacity monitoring and monitoring tools for network state. In short, while our survey is small, it helps the hypothesis that network administrators face complicated symptoms and causes.

4. ATPG Systems:

Depending on network model ATPG generates less number of test packets so that every forwarding rule is exercised and covered by at least one test packet. When an error is found, ATPG use different localization algorithm to ascertain the failing rules in network model.

5. Network Monitor:

To send and receive test data packet network monitor assumes special test agents in the network. The network monitor gets thedatabase and builds test packets and instructs each different to send the proper packets Recently test agents partition test packets by IP Proto field and TCP/UDP port number but other fields like IP option can be used If any tests fail the monitor chooses extra test packets from booked packets to find the faults The process gets repeated till the fault has been identified To communicate with test agents monitor uses and SQL it string matching to lookup test packets efficiently.

Performance:

The principal component overhead for ATPG are polling the network periodically for forwarding state and performing two reachable While one can reduce overhead by running the offline ATPG calculation less frequently this runs the risk of using out-ofdate forwarding information we reduce overhead in two ways First we have recently fast up the all-pairs reach ability calculation using a fast multithreaded. Second, instead of extracting the complete network state every time ATPG is triggered an incremental state updater can significantly reduce both the retrieval time and the time to calculate reach ability We are working on a real life version of ATPG that incorporates both techniques Test agents within terminals incur negligible overhead because they merely de multiplex test packets addressed to their IP address at a modest rate compared to the link speeds gb most modern CPUs are capable taken.

Conclusion:

In current System it uses a method that is neither exhaustive nor scalable different it reaches all pairs of edge nodes it could take detect faults in likeness properties ATPG goes much further than likeness testing with different framework ATPG could test for reach ability model and performance methods Our implementation also enlarges testing with simple errors localization scheme also build using header space framework

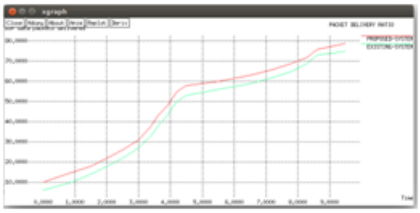
Future Enhancement:

Even one of the requirements gathered through the voice of customers and feedback different users are implemented there are always opportunities to enhance

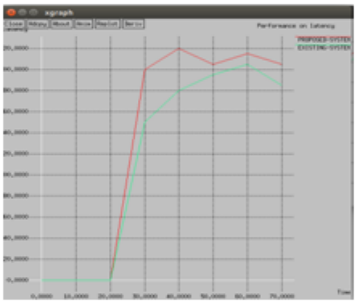
ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

model this tool and take it to the onelevel by automating different steps involved upon any level of code changes Explore automatically generating the unit tests results specific to the project without different the platform and save them to the output PDF Explore automatically generating the code coverage report and integrate in to the code review packet generation process Provide users used to upload the file directly to the given network location.

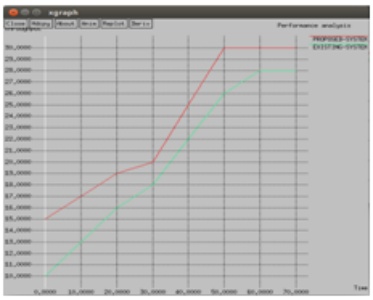
Results



Maximum no of packets delivered in proposed system



Performance of latency in proposed system



Performance analysis in proposed system

References:

[1] HongyiZeng, PeymanKazemian, George Varghese, ACM, and Nick McKeown, ACM,“Automatic Test Packet Generation”.

[2] M. Jain, 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.

[3] Kompella, R. R., Greenberg, A., Rexford, J., Snoeren, A. C., Yates, J. Cross-layer Visibility as a Service", In Proc. of fourth workshop on Hot Topics in Networks (HotNet-IV) (2005)

[4] D. Maltz, G. Xie, J. Zhan, H. Zhang, G. Hjalmtysson, A. Greenberg,"Routing design in operational networks: A look from the inside", In Proc. ACM SIGCOMM, 2004.

[5] Mark Stemm, Randy Katz, SrinivasanSeshan,“A network measurement architecture for adaptive applications”, In Proceedings of the nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 285 - 294, 2000.

[6] Verma, D.,"Simplifying Network Administration using Policy based Management".

[7] P. Yalagandula, P. Sharma, S. Banerjee, S. Basu, S.-J. Lee, “S3: A scalable sensing service for monitoring large networked systems".

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest,"Introduction To Algorithms", Eastern Economy Edition.New Delhi: Prentice-Hall of India Private Limited 1998.

[9] E Balagurusamy,"Programming in ANSI C", Second edition. New Delhi: Tata McGraw-Hill, 1997.

[10] Deitel&Deitel, C++ How To Program, Second Edition, New Jersey, NJ Prentice Hall, 1997

[11] Tom Christiansen, NathnTorkington,"Perl Cookbook: Solutions and Examples For Perl Programmers", 1st Edition, Sebastopol, CA: O’Reilly & Associates, Inc., 1999.

ISSN No: 2348-4845
International Journal & Magazine of Engineering, Technology, Management and Research
A Peer Reviewed Open Access International Journal

[12] Nathan Patwardhan, Ellen Siever and Stephen Spainhour, PERL In A Nutshell, 2nd Edition, Sebastopol, CA: O'Reilly & Associates, Inc., 2002.

[13] Tom Milligan, "Using Perl With Rational ClearCase Automation Library", 2004Retrieved Nov10, 2010 fromWorld Wide Web

[14] E. Dustin, J. Raska, J. Paul,"Automated Software Testing: Introduction Management and Performance", New York, NY: Addison-Wesley, 1999.

[15] R. S. Pressman,"Software Engineering: A Practitioner's Approach", 4th edition. Berkeley, CA: Osborne/McGrawHill, 1997.