ATPG: ATPG System in Fault Diagnosis System

Regonda Nagaraju  
Assistant Professor, Department of CSE,  
CMR Technical Campus, Kandlakoya Village,  
Medchal Mandal, Ranga Reddy District, Telangana,  
Email: nagcse01@gmail.com

M. Ajay Kumar  
Assistant Professor, Department of CSE,  
CMR Technical Campus, Kandlakoya Village,  
Medchal Mandal, Ranga Reddy District, Telangana,  
Email: ajayaminindla@gmail.com

Abstract – Now a day’s Networks are getting larger and more complex, hence network admin depend on normal tools such as ping and to trace route debug the problems. We are proposing automatic and systematic approach for testing and debugging networks called “Automatic Test Packet Generation and Fault Localization”. ATPG read router configurations and generates a unique model. This model is generating a minimum set of test packets to exercise every link in network exercise every rule in the network. Test packets are sent periodically and detected failure trigger a separate mechanism to localize the fault. ATPG can detect both functional testing and performance testing problems. ATPG complements but goes beyond earlier work in static checking or fault localization. We describe our prototype ATPG implementation and results on two real-world data sets applications: like Stanford University’s backbone network and Internet2. We find that small number of test packets suffices test all rules in these networks.


I. INTRODUCTION

It is popularly known us, very difficult to troubleshoot or identify and remove errors in networks. Every day, network engineers fight with mislabeled cables, software bugs, router misconfigurations, fiber cuts, faulty interfaces and other reasons that cause networks to drop down. Network engineers hunt down bugs with various tools (e.g., Ping, trace route, SNMP) and track down the reason for network failure using a combination of accrued wisdom and impression. a campus network may serve 50 000 users, a 100-Gbs long-haul link may carry 100 000 flows) and are getting complicated (with over 6000 RFCs, router software was based on millions of lines of source code, and network chips contain billions of gates. Fig. 1 is a simplified view of network state. Bottom of the figure is the forwarding state to forward each packet, consist of L2 and L3 forwarding information base (FIB), access control lists, etc. The forwarding state was written by the control plane (that could be local or remote) and should correctly implement the network administrator’s scheme. Examples of the scheme include: “Security group X was isolated from security Group Y,” “Use OSPF for routing,” and “Video traffic received at least 1 Mb/s.” We could think of the controller compiling the scheme (A) into device specific configuration files (B), which in turn determine the forwarding behavior of each packet (C). To ensure the network behave as designed, the three steps should remain consistent every times. Minimally, requires that sufficient links and nodes are working; the control plane identifies that a laptop can access a server, the required outcome can fail if links fail. The main reason for network failure is hardware and software failure, and this problem is recognized themselves as reachability failures and throughput/latency degradation. Our intention is to automatically find these kinds of failures. The intention of this paper is to generate a minimum set of packets automatically to cover every link in the network. This tool can automatically generate packets to test performance assertions like packet latency. ATPG detects errors independently and exhaustively testing forwarding entries and packet processing rules in network. In this tool, test packets are created algorithmically from the device configuration files and First information base, with minimum number of packets needed for complete coverage.

Test packets are fed into the network in which every rule was exercised directly from the data plan. Since ATPG treats links just like normal forwarding rules, the full coverage provides testing of every link in network. It could be particularized to generate a minimal set of packets that test every link for network liveness. For reacting to failures, many network operators like Internet proactively test the health of the network by pinging between all pairs of sources. Organizations can modify ATPG to face their needs; for example, they can select to test for network liveness (link cover) or test every rule (rule cover) to make sure security policy. ATPG could be modified to test reachability and performance. ATPG can adapt to constraints such as taking test packets from only a few places in the network or using particular routers to generate test packets from every port.

Fig. 1. Static versus dynamic checking: A scheme is compiled to forwarding state, and it is executed by the forwarding plane.
The contributions of this paper are as follows: 1) A survey of network operators exposing common failures and root causes. 2) A test packet generation algorithm. 3) A fault localization algorithm to separate faulty devices and Rules. 4) ATPG use cases for functional and throughput testing. 5) Evaluation of prototype ATPG system using rule sets gathered from the Stanford and Internet2 backbones.

II. METHODOLOGY

The proposed system can be divided into following modules:
A. Failures and root causes of network operators
B. Data plane analysis
C. Network troubleshooting
D. ATPG system
E. Network Monitor

A. 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.

B. 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 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

C. 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.

D. 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

E. Network Monitor:
To send and receive test data packet network monitor assumes special test agents in the network The network monitor gets the database 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.

![System Architecture](image1)

Fig.2. System Architecture

![Adding the nodes to network](image2)

Fig.3. Adding the nodes to network
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**