**Automatic Test Packet Generation**

**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.

**EXISTING SYSTEM:**

* 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.

**DISADVANTAGES OF EXISTING SYSTEM:**

* 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.

**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.

**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.

**SYSTEM ARCHITECTURE:**

****

**BLOCK DIAGRAM:**

N

N

N

Parser

All Pairs Reachability

Test Packet

DB

Test Packet Generator

Fault Localization

**MODULES:**

* Test Packet Generation
* Generate All-Pairs Reachability Table
* ATPG Tool
* Fault Localization

**MODULES DESCRIPTION:**

**Test Packet Generation:**

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 First Port (ATPG must only use test terminals that are available) and Header (ATPG must only use headers that each test terminal is permitted to send).

**Generate All-Pairs Reachability Table:**

ATPG starts by computing the complete set of packet headers that can be sent from each test terminal to every other test terminal. For each such header, ATPG finds the complete set of rules it exercises along the path. To do so, ATPG applies the all-pairs reachability algorithm described. On every terminal port, an all- header (a header that has all wild carded bits) is applied to the transfer function of the first switch connected to each test terminal. Header constraints are applied here.

**ATPG Tool:**

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.

**Fault Localization:**

ATPG periodically sends a set of test packets. If test packets fail, ATPG pinpoints the fault(s) that caused the problem. A rule fails if its observed behavior differs from its expected behavior. ATPG keeps track of where rules fail using a result function “Success” and “failure” depend on the nature of the rule: A forwarding rule fails if a test packet is not delivered to the intended output 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.

**SYSTEM REQUIREMENTS:**

**HARDWARE REQUIREMENTS:**

* System : Pentium IV 2.4 GHz.
* Hard Disk : 40 GB.
* Floppy Drive : 1.44 Mb.
* Monitor : 15 VGA Colour.
* Mouse : Logitech.
* Ram : 512 Mb.

**SOFTWARE REQUIREMENTS:**

* Operating system : Windows XP/7.
* Coding Language : JAVA
* IDE : Eclipse Kepler

**REFERENCE:**

Hongyi Zeng, Peyman Kazemian,George Varghese,and Nick McKeown,“**Automatic Test Packet Generation**”,VOL. 22, NO. 2, APRIL 2014.