Shanlax International Journal of Arts, Science and Humanities

Vol. 5 No. 2 October 2017 ISSN: 2321-788X UGC Approval No: 43960 Impact Factor: 2.114

RIHT: A NOVEL HYBRID IP TRACEBACK SCHEME

Article Particulars

Received: 24.10.2017 Accepted: 28.10.2017 Published: 30.10.2017

Prof.D.KIRUBASANKARI, M.Sc., M.Phil.,B.Ed.,
Head, PG and Research Department of Computer Science,
Sri Bharathi Women's Arts and Science College,
Kunnathu, Arni, TV Malai Dt, Tamil Nadu, India

U.GOMATHI
M.Phil Research Scholar, PG and Research Department of Computer Science,
Sri Bharathi Women's Arts and Science College,
Kunnathu, Arni, TV Malai Dt, Tamil Nadu, India

Abstract

It is long known attackers may use forged source IP address to conceal their real locations. To capture the spoofers, a number of IP traceback mechanisms have been proposed. How- ever, due to the challenges of deployment, there has been not a widely adopted IP traceback solution, at least at the Internet level. As a result, the mist on the locations of spoofers has never been dissipated till now. This paper proposes passive IP traceback (PIT) that bypasses the deployment difficulties of IP traceback techniques. PIT investigates Internet Control Message Protocol error messages (named path backscatter) triggered by spoofing traffic, and tracks the spoofers based on public available information (e.g., topology). In this way, PIT can find the spoofers without any deployment requirement. This paper illustrates the causes, collection, and the statistical results on path backscatter, demonstrates the processes and effectiveness of PIT, and shows the captured locations of spoofers through applying PIT on the path backscatter data set. These results can help further reveal IP spoofing, which has been studied for long but never well understood. Though PIT cannot work in all the spoofing attacks, it may be the most useful mechanism to trace spoofers before an Internet-level traceback system has been deployed in real.

Keywords: *IP traceback, PIT, Internet level traceback.*

Intoduction

It is notoriously hard to debug networks. Every day, network engineers wrestle with router misconfigurations, fiber cuts, faulty interfaces, mislabelled 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., Ping, trace route, tcpdump, 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). It is a mall wonder that network engineers have been labeled "masters of complexity".

In this paper we call an Automatic Test Packet Generation (ATPG) framework that automatically generates a minimal set of packets to test the live ness 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. In *Example 1*, instead of Alice manually deciding which ping packets to send, the tool does so periodically on her behalf. In *Example 2*, the tool determines that it must send packets with certain headers to "exercise" the video queue, and then determines that these packets are being dropped. 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. 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 proactively check the health of their network using pings between all pairs of sources. However, all-pairs ping does not guarantee testing of all links and has been found to be un-scalable for large networks such as Planet Lab. This paper is organized as. A survey of network operators revealing common failures and root causes Section II,A test packet generation algorithm Section III,A fault localization algorithm to isolate faulty devices and rules Section IV,ATPG use cases for functional and performance testing Section V, Evaluation of a prototype ATPG system using rule sets collected from the Stanford and Internet2 backbones Section VI, and Conclusion section VII.

Literature Survey

To send and receive test packets, network monitor assumes special test agents in the network. The network monitor gets the database and builds test packets and instructs each agent 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 problem. The process gets repeated till the fault has been identified. To communicate with test agents, monitor uses JSON, and SQ Lite's string matching to lookup test packets efficiently ATPG uses the header space framework—a geometric model of how packets are processed we described in. In header space, protocol-specific meanings associated with headers are ignored: A header is viewed as a flat sequence of ones and zeros. A header is a point (and a flow is a region) in the space, where is an upper bound on header length. By using the header space framework, we obtain a unified, vendor-independent, and protocol-agnostic model of the network2 that simplifies the packet generation process significantly. 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

A. Network Model

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.

B. 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.

C. Data 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.

D. 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 the ideal tool for network debugging it is 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.

Algorithms and Techniques Used for Traceback Scheme

Our goal is to come up with a collection of take a look at packets to exercise each rule each switch perform, in order that any fault are determined by a minimum of one take a look at packet. The broader goal will be restricted to testing each link or each queue. Once generating take a look at packets, ATPG should respect 2 key constraints:

a) Port: ATPG should solely use take a look at terminals that square measure available;

b) Header: ATPG should solely use headers that every take a look at terminal is allowable to transfer. for instance, the network administrator might solely enable employing a specific set of VLANs.

A. Fault Localization Algorithm

Given a list of (pk0, (R(pk0)), (pk1, (R(pk1)) …(pkn,(R(pkn)) tuples, find all that satisfies ∫pki,R(pki,r)=0.

There are three steps used in fault localization algorithm,

Step 1: Consider the results from sending the regular test packets. For every passing test, 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 assumption, one or more of the rules *F* are in error. Therefore *F-P*, is a set of suspect rules.

Step 2: ATPG next time the set of suspect rules by weeding out correctly working rules. ATPG does this using the *reserved packets* (the packets eliminated by Min-Set-Cover). ATPG selects reserved packets whose rule histories contain exactly one rule from the suspect set and sends these packets. Suppose a reserved packet *p* exercises only 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 removed from the suspect set. ATPG repeats this process for each reserved packet chosen in Step 2.

Step 3: In most cases, the suspect set is small enough after Step 2, which 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.

B. PODEM Algorithm

PODEM (Path-Oriented Decision Making) is an Automatic Test Pattern Generation (ATPG) algorithm which was created to overcome the inability of D-Algorithm (D-ALG) to generate test vectors for circuits involving Error Correction and Translation. The aim of this project is to implement the PODEM algorithm to generate test vectors for a given fault. External tools such as HITEC/PROOFS package are used to convert a netlist of a circuit into a level zed circuit description. HITEC/PROOFS package is also used to calculate the Testability Measures required for implementation of PODEM. A sample circuit is chosen for verification purposes. Various subroutines of the PODEM algorithm are individually verified. Finally the test vectors generated by the program are compared with manual implementation of the PODEM algorithm.

PODEM proves to be more efficient as compared to a D-ALG because it limits its search space only to Primary Inputs (PIs) of the circuits. D-ALG on the other hand has a search space comprising of all the internal nodes of the circuit along with the PIs. The first objective of the algorithm is to sensitize the fault. After the fault is sensitized the objectives are changed in order to propagate the fault to a Primary Output (PO). Function OBJECTIVE is used to determine objectives for the program. Depending on the current objective, a function called BACKTRACE is used to determine the value of one of the PIs. For every PI assigned, logic simulation is performed to check for two conditions: desensitization of the fault and disappearance of fault propagation path (also known as X-PATH CHECK). If any one of the two conditions is violated, the program backtracks and Changes the value assigned to the most recent PI. This process of assigning values to PIs is repeated till PIs form a test vector or no more combinations of PIs are possible. The latter case implies that the test is untestable.

The simplified flowchart and its major functions Objective are shown in the following flowchart.

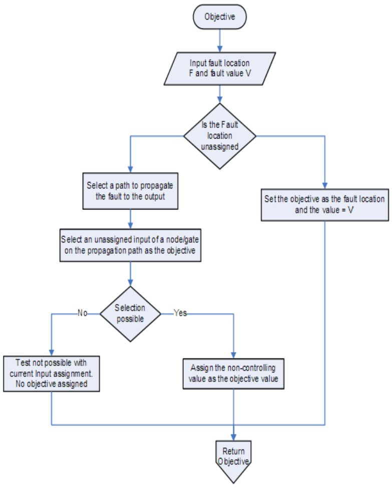

Objective
Input fault location F and fault value V
Is the Fault location unassigned
Select a path to propagate the fault to the output
Set the objective as the fault location and the value = V
Select an unassigned input of a node/gate on the propagation path as the objective
Selection possible
No
Yes
Test not possible with current Input assignment. No objective assigned
Assign the non-controlling value as the objective value
Return Objective

Fig.1 PODEM objective

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.

```
Function $T_i(p_k)$
    # Iterate according to priority in switch i
        For r € $ruleset_i$ do
          For pk € $r._{matchset}$ then
              $Pk._{history}$ <- $Pk._{history}$ U{r}
        Return $r(p_k)$
    Return {(drop, $p_k$.h)]
```

A. ATPG Methods and Algorithm

- ATPG enables testers to distinguish between the correct circuit behavior and the faulty circuit behavior based on inputs patters
- The generated patterns are used to test semiconductor devices for defects after manufacturing
- A defect is an error introduced into a device during the manufacturing process
- The effectiveness of ATPG is measured by the amount of modeled defects, or fault models, that are detected and the number of generated patterns.
- The effectiveness of ATGP gives an estimate of test quality
- A fault model is a mathematical description of how a defect alters design behavior
- A fault is aid to be detected by a test pattern if, the faulty circuit output differs from the original circuit output

There are two steps that ATPG should take to detect fault: i) Fault activation, ii) Fault Propagation.

B. ATPG: D-Algorithm

- An error is observed due to differing values at a line in the circuit with or without failure. Such a divergence is denoted by values D or D´ to mark differences 1/0 or 0/1, respectively.
- Instead of Boolean values, the set {0, 1, D, D´} is used to evaluate gates and carry out implications.
- A gate that is not on a path between the error and any output does never have a D-value.
- A necessary condition for testability is the existence of a path from the error to an output, where all intermediate gates either have a D-value or are not assigned yet.
- A gate is on a D-chain, if it is on a path from the error location to an output and
- All intermediate gates have D-values.

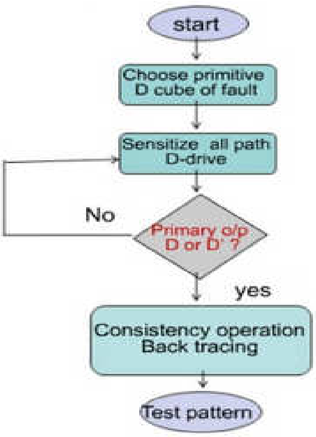
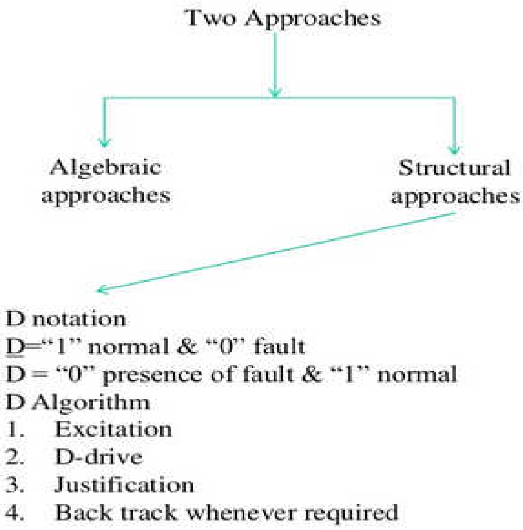

start
Choose primitive D cube of fault
Sensitize all path D-drive
No
Primary o/p D or D' ?
yes
Consistency operation Back tracing
Test pattern

Fig 2. D-Algorithm



D notation

D=“1” normal & “0” fault

D = “0” presence of fault & “1” normal

D Algorithm

1. Excitation
2. D-drive
3. Justification
4. Back track whenever required

Fig 3. Structure of D-Algorithm

i) D-Algorithm – Roth

Roth's D-Algorithm (D-ALG) defined the calculus and algorithms for ATPG using D-cubes.

Definitions

Singular cover: Defined to be the minimal set of input signal assignments needed to rep-resent essential prime implicants in Karnaugh map

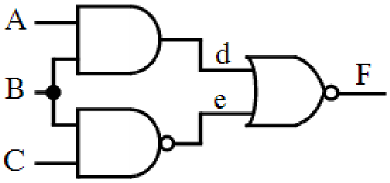


Fig 4.Karnaugh Process

C. ATPG Algorithm Types

i) Exhaustive Algorithm: For n-input circuit, generate all 2n input.Infeasible, unless circuit is partitioned into cones for logic, with < 15 inputs.

ii) Random Pattern Generation: Used to get tests for 60-80% of the faults. The D-algorithm or other ATPG algorithms used for the rest. Fault simulation is necessary in order to select useful patterns. Weighted random patterns: 0 and 1 are not equally likely.

Symbol	Roth's algebra		Muth's algebra	
	Good	Failing	Good	Failing
D	1	0	1	0
D	0	1	0	1
0	0	0	0	0
1	1	1	1	1
X	X	-	X	X
G0	-	-	0	X
G1	-	-	1	X
F0	-	-	X	0
F1	-	-	X	1

ATPG Algorithm Types

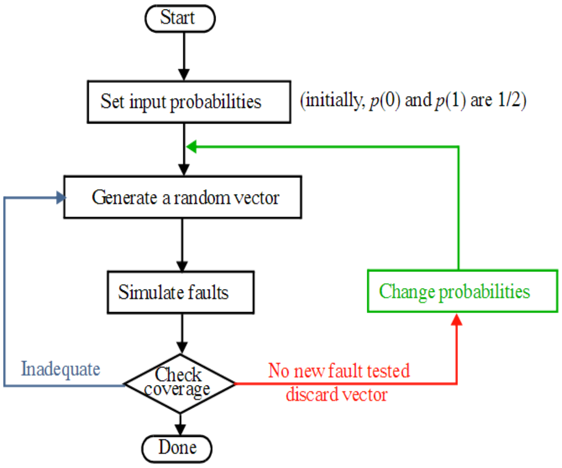


Fig 5. RPG Method

D. Origination of Test Packets:

The ATPG system can be roughly divided into two parts namely test packet generation and fault localization. While developing an algorithm for test packet generation a supposition is that, set of test terminals may transmit or take in test packets. The target for algorithm is generating minimum number of test packets to practice every rule in every switch function, as a result if a fault occurs, it will be watched by at least one test packet. ATPG system makes use of test packets selection algorithm (TPS) to generate test packets. ATPG must only make use of test terminals that are available and ATPG must utilize headers that each test terminal is authorized to send are two important restrictions of which ATPG must take a notice of at the time of generating test packets.

1) ATPG system begins by estimating entire set of test packet headers that can be forwarded from each test terminal to every other test terminal. ATPG achieves this by detecting full set of rules it can work out in entire journey. Thus, ATPG refers to all pair reach ability algorithm to perform this task.

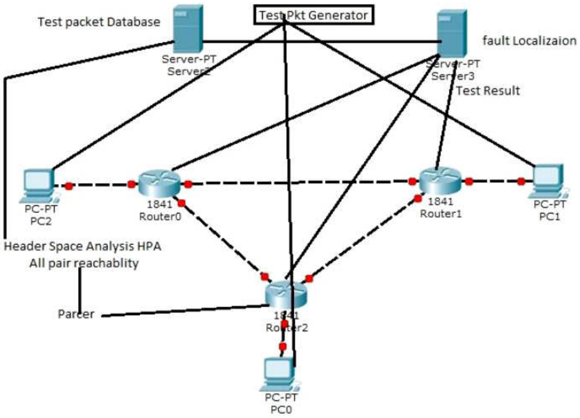


Fig 6. Automatic Test Packet Generation

Networking

Networking is the word basically relating to computers and their connectivity. The networks between the computing devices are very common these days due to the launch of various hardware and computer software which aid in making the activity more convenient to build and use.

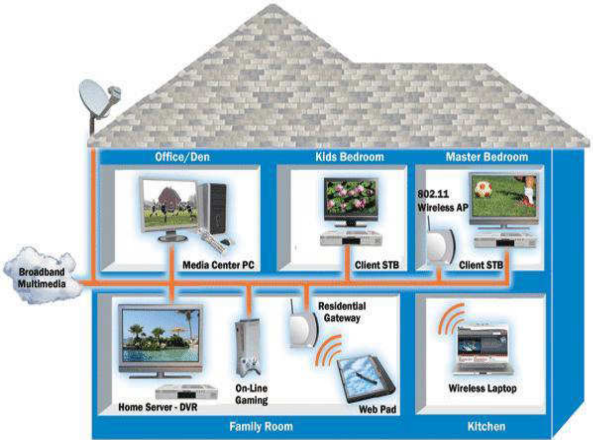
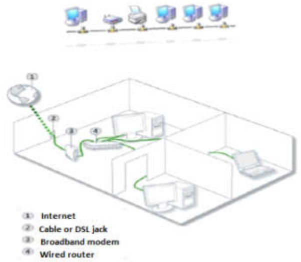


Fig 7. Structure of Networking

A. Networking Functions



When computers communicate on a network, they send out data packets without knowing if anyone is listening. Computers in a network all have a connection to the network and that is called to be connected to a network bus. What one computer sends out will reach all the other computers on the local network. Above diagrams show the clear idea about the networking functions. For the different computers to be able to distinguish between each other, every computer has a unique ID called MAC-address (Media Access Control Address). This address is not only unique on your network but unique for all devices that can be hooked up to a network. The MAC-address is tied to the hardware and has nothing to do with IP-addresses. Since all computers on the network receives everything that is sent out from all other computers the MAC-addresses is primarily used by the computers to filter out incoming network traffic that is addressed to the individual computer. When a computer communicates with another computer on the network, it sends out both the other computers MAC-address and the MAC-address of its own. In that way the receiving computer will not only recognize that this packet is for me but also, who sent this data packet so a return response can be sent to the sender. MAC-address (Media Access Control Address) This address is not only unique on a network but unique for all devices that can be hooked up to a network. The MAC-address is tied to the hardware and has nothing to do with IP-addresses. Since all computers on the network receives everything that is sent out from all other computers the MAC-addresses is primarily used by the computers to filter out incoming network traffic that is addressed to the individual computer. When a computer communicates with another computer on the network, it sends out both the other computers MAC-address and the MAC-address of its own. In that way the receiving computer will not only recognize that this packet is for me but also who sent this data packet so a return response can be sent to the sender.

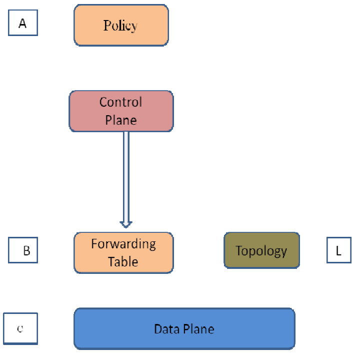


Fig 8. Network state

The above Figure network state can be decomposed in three parts as A, B and C. We can consider the policy (A), which is compiled by controller into configuration files which are device specific (B), which then shows the forwarding behavior of every packet (C). To ensure the network behaves as per requirement, all the three steps at all times should remain consistent, that is same as A=B=C. At the same time, the topology, shown at the bottom right in the figure, should also be able to satisfy a set of liveness properties shown by L.

B. Types of Network

Organizations of different structures, sizes, and budgets need different types of networks. Networks can be divided into one of two categories:

1. Peer-to-Peer Network
2. Server-Based Networks
3. Network Communications

C. Advantages of Network

1. Easy Communication
2. Ability to Share Files, Data and Information
3. Sharing Hardware
4. Sharing Software
5. Security
6. Speed

D. 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).

E. Network Design

As mentioned in the last section, the automatic test packet generation (ATPG) system makes use of geometric model of header space analysis. This section explains some of the key terms associated with geometric framework of header space analysis

1. *Packet*

Packet in a network can be described as a tuple of the form (port, header).Each one of the port is allotted with one and only one unique number.

2. *Switch*

Another term used in geometric model of header space analysis is switches. It is the job of switch transfer Function T, to model devices in a network. Example of devices can be switches or routers. Each incoming packet is coupled with exactly single rule.

3. *Rules*

Piece of work for rules is generation of list of one or more output packets associated with those output ports to which the packet is transferred, and explain how fields of port are modified. In other words, rules explains how the region of header space at entrance in changed into region of header space at exit

4. *Topology*

The network topology is modeled by topology transfer function. The topology transfer function gives the specification about which two ports are joined by links.

5. *Life of a Packet*

One can see life of a packet as carrying out or executing switch transfer function and topology transfer function at length. When a particular packet comes in a network port p, firstly a switch function is applied to that packet. Switch transfer function also contains input port pk.p of that packet. The result of applying switch function is list of new packets [pk1, pk2, pk3,].

Experimental Evaluation

A. Network Simulator (NS2)

Simulation can be defined as "Imitating or estimating how events might occur in areal situation". It can involve complex mathematical modeling, role playing without the aid of technology, or combinations. The value lies in the pacing you under realistic conditions that change as a result of behaviour of others involved, so you cannot anticipate the sequence of events or the final outcome.

i) NS2 Overview

NS is an event driven network simulator developed at University of California at Berkeley, USA, as a REAL network simulator projects in 1989 and was developed at with cooperation of several organizations. Now, it is a VINT project supported by DARPA.NS is not a finished tool that can manage all kinds of network model. It is actually still anon-going effort of research and development. The users are responsible to verify that their network model simulation does not contain any bugs and the community should share their discovery with all. There is a manual called NS manual for user guidance.

NS is a discrete event network simulator where the timing of events is maintained by a scheduler and able to simulate various types of network such as LAN and WPAN according to the programming scripts written by the user. Besides that, it also implements variety of applications, protocols such as TCP and UDP, network elements such as signal strength, traffic models such as FTP and CBR, router queue management mechanisms such as Drop Tail and many more.

There are two languages used in NS2 C++ and OTcl (an object oriented extension of Tcl). The compiled C++ programming hierarchy makes the simulation efficient and execution times faster. The OTcl script which written by the users the network models with their own specific topology, protocols and all requirements need. The form of output produce by the simulator also can be set using OTcl.

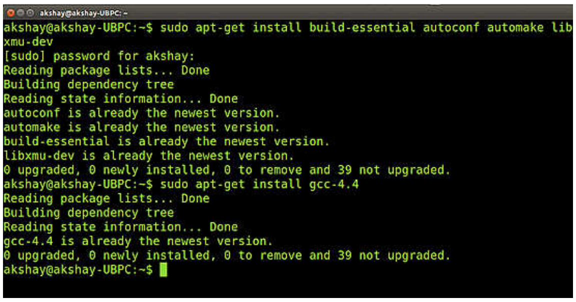
ii) Building the Dependencies

Ns2 requires a few packages to be pre installed. It also requires the GCC- version 4.3 to work correctly. So install all of them by using the following command:

# sudoapt-get install build-essential auto confautomakelibxmu-dev

#sudo apt-get install gcc-4.4

The image below shows the output of executing both the above commands. If you have all the dependencies pre-installed, as I did, the output will look like the image below:



```
akshay@akshay-UBPC: ~
akshay@akshay-UBPC:~$ sudo apt-get install build-essential autoconf automake lib
xmu-dev
[sudo] password for akshay:
Reading package lists... Done
Building dependency tree
Reading state information... Done
autoconf is already the newest version.
automake is already the newest version.
build-essential is already the newest version.
libxmu-dev is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 39 not upgraded.
akshay@akshay-UBPC:~$ sudo apt-get install gcc-4.4
Reading package lists... Done
Building dependency tree
Reading state information... Done
gcc-4.4 is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 39 not upgraded.
akshay@akshay-UBPC:~$
```

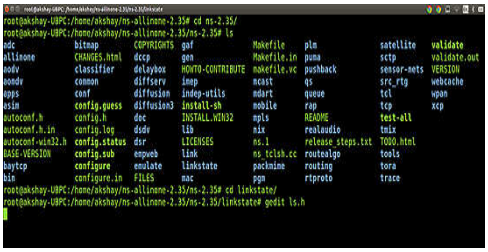
Fig 9. NS2 installation

Navigate to the folder "link state", use the following command. Here it is assumed that the ns folder extracted is in the home folder of your system.

#cd ~/ns-allinone-2.35/ns-2.35/link state

Now open the file named "ls.h" and scroll to the 137th line. In that change the word **"error"** to **"this->error"**. The image below shows the line 137 (highlighted in the image below) after making the changes to the ls.h file. To open the file use the following command:

geditls.h



```
root@akshay-UBPC: /home/akshay/ns-allinone-2.35/ns-2.35/linkstate
root@akshay-UBPC:/home/akshay/ns-allinone-2.35# cd ns-2.35/
root@akshay-UBPC:/home/akshay/ns-allinone-2.35/ns-2.35# ls
adc              bitmap         COPYRIGHTS  gaf                Makefile      plm                satellite    validate
allinone         CHANGES.html   dccp        gen                Makefile.in   puma               sctp         validate.out
aodv             classifier     delaybox    HOWTO-CONTRIBUTE   makefile.vc   pushback           sensor-nets  VERSION
aomdv            common         diffserv    imep               mcast         qs                 src_rtg      webcache
apps             conf           diffusion   indep-utils        mdart         queue              tcl          wpan
asim             config.guess   diffusion3  install-sh         mobile        rap                tcp          xcp
autoconf.h       config.h       doc         INSTALL.WIN32      mpls          README             test-all
autoconf.h.in    config.log     dsdv        lib                nix           realaudio          tmix
autoconf-win32.h config.status  dsr         LICENSES           ns.1          release_steps.txt  TODO.html
BASE-VERSION     config.sub     empweb      link               ns_tclsh.cc   routealgo          tools
baytcp           configure      emulate     linkstate          packmime      routing            tora
bin              configure.in   FILES       mac                pgm           rtproto            trace
root@akshay-UBPC:/home/akshay/ns-allinone-2.35/ns-2.35# cd linkstate/
root@akshay-UBPC:/home/akshay/ns-allinone-2.35/ns-2.35/linkstate# gedit ls.h
```

Fig 10. Save and Close process of NS2

B. Result and Discussion

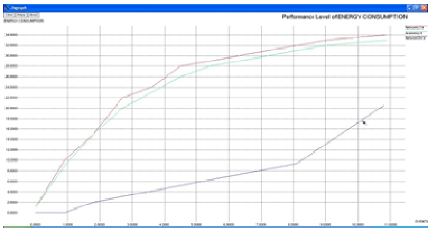


Fig 11. Node Creation

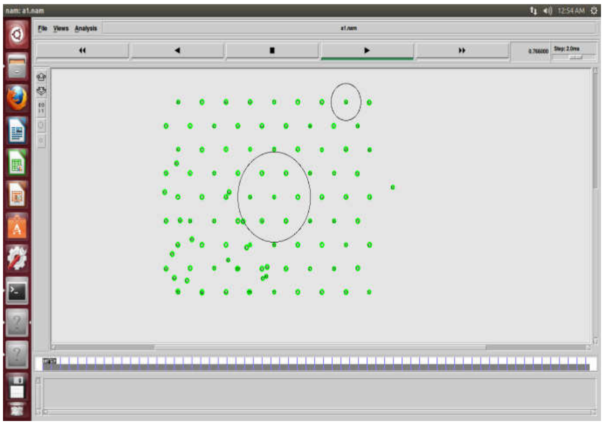


Fig 12. Automatic Packet filtering

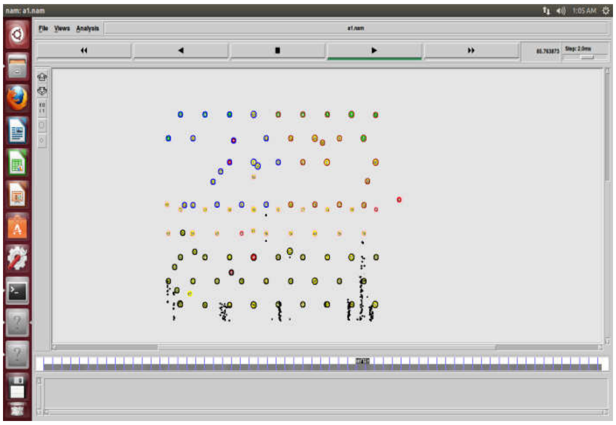


Fig 13. Energy Consumption

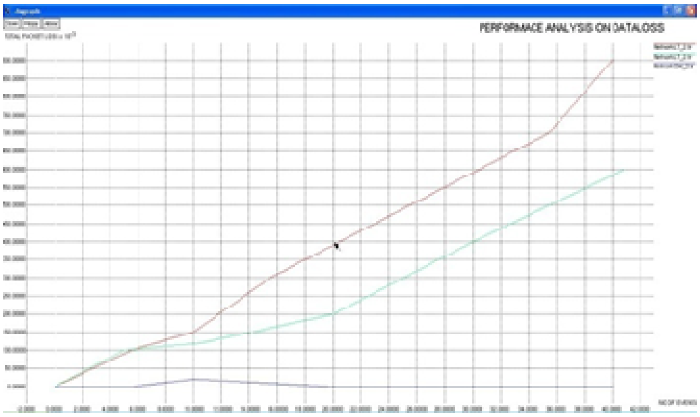


Fig 14. Packet Loss

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 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 reach ability), 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 reach ability 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 (40,000 for Stanford, and 40000 for Internet2).

Network managers today use primitive tools such as and trace route. Our survey results indicate that they are eager for more sophisticated tools. In fact, many months after we built and named our system, we discovered to our surprise that ATPGwas awell-known acronym in hardware chip testing, where it stands for Automatic Test Pattern Generation. We hope network ATPG will be equally useful for automated dynamic testing of production networks.

References

1. “ATPG code repository,” [Online]. Available: http://eastzone.github. com/atpg/
2. “Automatic Test Pattern Generation,” 2013 [Online]. Available: http://en.wikipedia.org/wiki/Automatic_test_pattern_generation
3. P. Barford, N. Duffield, A. Ron, and J. Sommers, “Network performance anomaly detection and localization,” in Proc. IEEE INFOCOM, Apr. , pp. 1377–1385.
4. “Beacon,” [Online]. Available: http://www.beaconcontroller.net/
5. 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.
6. C. Cadar, D. Dunbar, and D. Engler, “Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs,” in Proc. OSDI, Berkeley, CA, USA, 2008, pp. 209–224.
7. M. Canini,D.Venzano, P. Peresini,D.Kostic, and J. Rexford, “A NICE way to test Open Flow applications,” in Proc. NSDI, 2012, pp. 10–10.
8. A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot, “Netdiagnoser: Troubleshooting network unreachabilities using end-to-end probes and routing data,” in Proc. ACM CoNEXT, 2007, pp. 18:1–18:12..
9. N. Duffield, “Network tomography of binary network performance characteristics,” IEEE Trans. Inf. Theory, vol. 52, no. 12, pp. 5373–5388, Dec. 2006.

10. N. Duffield, F. L. Presti, V. Paxson, and D. Towsley, "Inferring link loss using striped unicast probes," in Proc. IEEE INFOCOM, 2001, vol. 2, pp. 915–923.
11. N. G. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," IEEE/ACM Trans. Netw., vol. 9, no. 3, pp. 280–292, Jun. 2001.
12. P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in Proc. ACM SIGCOMM, 2011, pp. 350–361.
13. "Hassel, the Header Space Library," [Online]. Available: https://bitbucket. org/peymank/hassel-public/
14. Internet2, Ann Arbor, MI, USA, "The Internet2 observatory data collections," [Online]. Available: http://www.internet2.edu/observatory/ archive/data-collections.html
15. M. Jain and 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.
16. P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in Proc. NSDI, 2012, pp. 9–9.
17. R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, "IP fault localization via risk modeling," in Proc. NSDI, Berkeley, CA, USA, 2005, vol. 2, pp. 57–70.
18. M. Kuzniar, P. Peresini, M. Canini, D. Venzano, and D. Kostic, "A SOFT way for OpenFlow switch interoperability testing," in Proc. ACM CoNEXT, 2012, pp. 265–276.