Automatic Troubleshooting of Network using Test Packet Generation

Udaysingh Mohan Bhosale [1], Prof. Amrit Priyadarshi [2]
[1] *Department of Information Technology, DGOI,FOE, Daund, Savitribai Phule Pune University,*
[2] *Department of Computer Engineering, DGOI,FOE, Daund, Savitribai Phule Pune University,*
Pune,Maharashtra, India.
[1]bhosale.udaysingh@gmail.com
[2] amritpriyadarshi@gmail.com

Abstract *As compared to early days of networks, today's network grows very rapidly. As network grows the network problems also grows. To test and debug the large network is the difficult and this is the problem for the network administrators and the network engineers. To check the liveness of network administrators use the traditional tools such as ping, trace route etc. But these tools have some drawbacks while troubleshooting the network. These drawbacks are overcome by the proposed system that is "Automatic testing and troubleshooting a network system". This approach generates a device independent model by getting router configurations. In this approach minimal set of test packets are generated and send to each and every link in a network and localize the fault if fault occurs.*

Keywords — *Test packet Generation, Fault Localization, Selection of Test Packet, Reachability table. Switch Transfer Function.*

I. INTRODUCTION

Today's network is very huge and complex and hence to monitor the network is become a difficult task for network administrators. Because of high scalability of network it contain software as well as hardware errors such as mislabelled cables, software bugs, router misconfigurations, fibre cuts, faulty interfaces and many more. These are the causes of network failure. Network administrators use tools such as ping, trace route etc. to troubleshoot the network. But using such tools it is necessary to provide manual entries for checking the liveness of network. In this way troubleshooting the network is very difficult because to send the packets routers use forwarding tables, forwarding rules, their policies and other configuration parameters in a distributed network. To observe the forwarding state of each and every router it is necessary to log into the each and every box. Also our network contains many sub different networks and so many networks have different policies .Sometime configuration parameters also changes and policies will be changed at particular instance while troubleshooting. Manually send the test packets to check aliveness of the network; the proposed system does so automatically. To overcome the difficulties that is hardware failure and software bugs the proposed system that is "Automatic troubleshooting of network using test packet generation" automatically create a less number of packets to test the aliveness of networks. The proposed system also automatically creates packets to test performance of a network. In existing system Administrator The proposed system will send generated packets periodically in the network and check each and every rule in the network. If fault is occurred then fault is localized and corrective actions will be implemented to solve the problem.

Troubleshooting a network is difficult for three reasons [1]. First, the forwarding state is distributed across multiple routers and firewalls and is defined by their forwarding tables, filter rules, and other configuration parameters. Second, the forwarding state is hard to observe because it typically requires manually logging into every box in the network. Third, there are many different programs, protocols, and humans updating the forwarding state simultaneously (SeeFig.1). But creating a tool using TPG algorithm would automate the entire process.

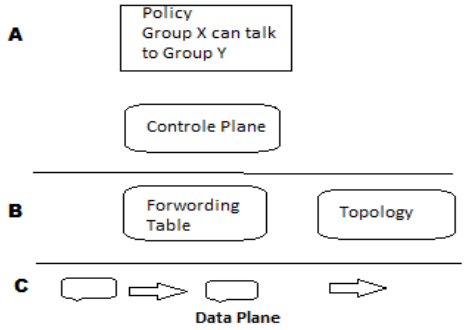


Fig.1.Static versus dynamic checking: A policy is compiles forwarding state, which is the n executed by the forwarding plane. Static checking (e.g., confirms that A=B. Dynamic checking (e.g., APTG in this paper) confirms that the topology is meeting liveness properties (L) and that B=C.

Fig.1 reveals the various views of network states. At the top of figure policy A which is to be compiled into the device specific configuration files i.e. into policy B. After compilation process the result is then decide the forwarding behaviour of the

policy C. This can be done by using control plane because forwarding state is written by the control plane. At the bottom of figure the forwarding state is used. The forwarding state is responsible for the movement of packets from source to its intended destination which contains FIB (Forwarding Information Base), ACL (Access Control Lists) etc. If you want to execute your network as per design then it is necessary that three forwarding states should remain equal at all the times.i.e.A=B=C.

Therefore our goal is "To build a system which would automatically monitor functional and performance faults in network. To detect and diagnose errors by independently and exhaustively testing all forwarding entries, firewall rules, and any packet processing rules in the network. To check the liveness and fault localization of the network.

II. RELATED WORK

A system which can read the router configuration and depending upon that information they design independent model known as ATPG [1] (Automatic Test Packet Generation).This system is very scalable and enhanced. They use modules that can be used to generate test packets and send them periodically to the network. The algorithm they used for that can check each and every link. While doing this the state information is send back to the system. If unexpected behaviour is occurred then the algorithm that is Fault localization algorithm will be invoked and handle the fault [1]. This system can also detect the performance and functional problems. They use this system on two large network Stanford University's backbone network and Internet2 and found that very small numbers of test packets are required to test the network. To cover all the network 4000 packets are required but for Stanford network they seen only 54 packets were cover the entire network.

A system Network performance anomaly detection and localization present a framework that contains three algorithms [2]. One algorithm is used to detect unnatural behaviour of paths in networks. Second selects set of links from source to destination if unnatural behaviour is occurred. The third algorithm is used to detect the root cause of unnatural behaviour. It means that the algorithm localize the fault or cause. The path selection algorithm is used to frequently monitor all the links and detect the unnatural behaviour of network and also minimize the link investigation overhead. After the completion of monitoring task the path information is generated. This path information is then used to localize the fault instead of additional investigation for searching the links associated between nodes. Hence it removes additional investigation overhead automatically. The authors proposed system reveals that as compared to previously existing system there system can maintain accuracy and increase performance with respect to time for detecting and localizing the unnatural behaviour of network.

The Robust monitoring of link delays and faults in IP networks was developed in [3]which was very flexible or adaptable system to supervise the errors occurred in ISP (Internet Service Provider) and the link problems in EIP (Enterprise IP) network. They used two phase approaches. In first phase they use less number of supervising stations. These stations are used to cover all the associated links between nodes even if links are failed it doesn't matter. In second phase parallel they execute stations that sends the inquiry messages to the network that can determines the network faults. For station (minimum set of stations) selection problems and link inquiry problems they used greedy approximation algorithm. This algorithm generates logarithmic approximation value for station selection problem. The algorithm also generates constant values for link assignment problems.

Unassisted and automatic generation of high-coverage tests are also used in [4] for complex systems program.They used a symbolic and systematic tool known as KLEE. The ability of this tool is to automatically generate tests. These tests obtain high scope on a complicated and comphrensive set of programs. They used KLEE and checked all 89 standalone programs in a GNU CORE UTILS utility suit which have core user level environment which was installed on several UNIX systems. As UNIX is open source system and hence it contain several tested programs which are available to all. KLEE generated test obtain high line scope that is an average of 90% per tools. When they did same for 75 equivalent tools in the embedded system that is on BUSY BOX system suit they obtain 100% coverage. On 31 of them they used the KLEE tool for finding the bugs. They used KLEE tool on approximately 452 applications and found that about 56 of them have serious bugs. In that out of 56 the 3 bugs were in CORE UTILS that had been missed from 15 years. Finally they used both BUSY BOX and CORE UTILS utilities for checking the functionality errors and thousands of deviations.

After the invention of Open Flow Capable Switches people used their functionality and their exciting features. But this is less reliable due to the programing errors. To reduce errors the centralized programing model. This model uses single program for network management. This model reveals less probability of errors. Authors specify a systematic and efficient technique for controlling unmodified programs known as NICE tool[5]. This tool is used to check the entire network space that is the controller, the switches and the hosts information. As there are so many challenges in the network such as scalability, data packet discovery, maintaining state information and event ordering. Authors address this by implementing an innovative and enhanced system that can check the symbolic execution of event handlers. Authors also present a

simplified Open Flow switch model to minimize the bugs by applying specific strategies.

III. NETWORK MODEL

Let's get familiar with some keywords.

Packets: A packet is defined by (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 T, 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

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 IPaddress); 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 [1].

Rule History: At any point, each packet has a rule history (r0, r1…) an ordered list of rules the packet matched so far as it traversed the network. Rule histories are fundamental to proposed system, as they provide the basic raw material from which systems constructs tests.

Topology: The topology transfer function, models the network topology by specifying which pairs of ports (psrc, pdst) 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.

Switch Transfer Function(STF) Pseudo Code:

```
STF Ti (Pk)
# Repeat upon priority of switch
for rule r  member of ruleseti
do
        If(Pk € r.matchset
                Pk.history←U{r}
                return Pk
        else
                return [(drop, Pk.h)]
end
```

IV. IMPLEMENTATION DETAILS

A. SYSTEM ARCHITCTURE

Current system –

The administrator manually decides which ping packets to send. Here, the approaches designed can prevent software logic errors but fails to detect failures caused by failed links and routers.

Proposed System-

Instead of the administrator, the system tool would do so periodically on his or her behalf. Whereas here, automatic troubleshooting of network using packet generation system automatically detects the failures by testing the liveness of the underlying topology.

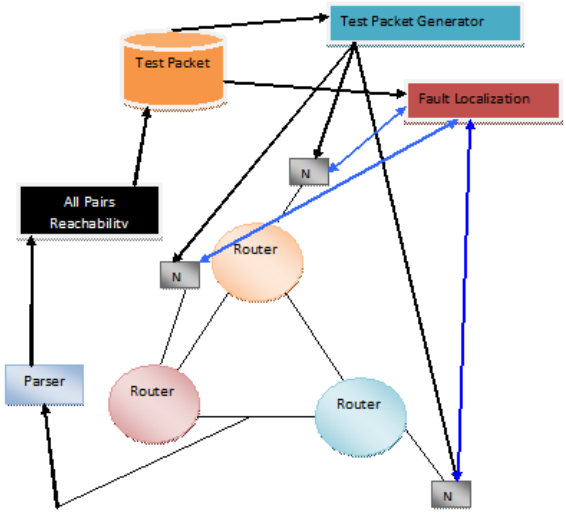


Fig 2. System Architecture

Steps of execution of proposed system-

1. The system first collects all the forwarding state from the network
2. System uses Header Space Analysis to compute reachability between all the test terminals.
3. The result is then used by the test packet selection algorithm to compute a minimal set of test packets that can test all rules.
4. These packets will be sent periodically by the test terminals.
5. If an error is detected, the fault localization algorithm is invoked to narrow down the cause of the error.

B. MODULES

Module 1: Generation of Test Packets:

The goal of this module is to develop a bunch of test packets to discover the nodes located in network. The test packet is provided to the switch or router. The router contains forwarded rules and hence the packets are forwarded in the network. In this way each and every link will be examined and the failure link will find out if packet is not delivered to intended port.

Module2: Generation of All-Pairs Reachability Table:

This module is used to compute the set of packet headers that can sent across the network. For that header, proposed module sends the set of rules it travelled along the path. For that purpose the all-pairs reachability algorithm is used .For every host

all header fields are applied with respect to the transfer function to the first terminal where you can send test packets. To do so, system applies the all-pairs reachability algorithm as follows:

1. Header constraints are applied. For example, if traffic can be sent on VLAN A, then instead of starting with an all- x header, the VLAN tag bits are set to A.

2. Set of rules that match the packet are recorded in packet history. Hence all-pairs reachability table as shown in table1.

Table 1: All pair Reachability table-Contains All headers from every node with their rule history

Header	Inport	Outport	Rule History
H_1	P_{11}	P_{12}	$\{r_{11},r_{12}\ldots..\}$
H_2	P_{21}	P_{22}	$\{r_{21},r_{22}\ldots..\}$
.	.	.	.
.	.	.	.
.	.	.	.
H_n	P_{n1}	P_{n2}	$\{r_{n1},r_{n2}\ldots..\}$

Module3: System Tool:

This module is used *to* generates the less number of test packets so that entire network is covered by at least one test packet. During this process if error is detected, this tool uses a fault localization algorithm to determine the failed links.

Module 4: Fault Localization:

The proposed system periodically sends test packets. If test packets is not reached or failed, the proposed system recognizes the fault and the root cause of that problem. Fault occurred if its observed behaviour differs from its expected behaviour. If a test packet is not delivered to the intended output port forwarding rule is failed, when packets are dropped drop rule is worked. When topology problems are there then we can say that this is the link failure.

Fault Model: A rule fails if its observed behaviour differs from its expected behaviour. Proposed system keeps track of where rules fail using a result function. For a rule, the result function is defined as

$R(r, P_{k)=}$ { 0, if packet P_k fails at rule i}
{1, if packet P_k succeeds at rule i}

V. MATHEMATICAL MODEL AND ALGORITHM:

S={s, e, i, o, f, DD, NDD, Succ, Fail}
s=header space analysis.
e=test packet generate and localize fault.
I=(I1,I2,I3. . . ..)
The deterministic data are,
I1=(1. . . .n) n=minimal set test packets.
The non-deterministic data are.
I1=(1. . . n) n= incorrect firewall rule
O=(O1)(output))
O1=(test the live ness of the underlying topology)
F=function Network(packets, switches, T)
For pk0 € packets do
T←FIND_SWITCH (pk0.p, switches)
For pk1 € T(pk0) do
If pk1.p € EdgePort then
#Reached edge
RECORD(pk1)
Else
#Find next hop
NETWORK(T(pk1),switches,T)
Succ=desired output is generated.
Fail=desired output is not generated.
For your reference
S- is system
s- start state of project.
e- end state of project.
i-input for your project
f-is functions
DD-deterministic data(means valid input data)
NDD-non deterministic data(invalid input data).
Succ- successful state of your project
Fail-failure state of your project

Algorithm:
INPUT: N1, N2, N3, R1, R2, R3, ATPG TOOL

START:

1: Packet PK arrives at a network port P.

2: The switch function that T contains the input port PK.P

3: Produce a list of packets.

4: If packet reaches destination it is ecorded.

ELSE Topology function called switch function containing new port.

5: Process repeats until packet reaches to destination or dropped.

END

OUTPUT: Packets reached status.

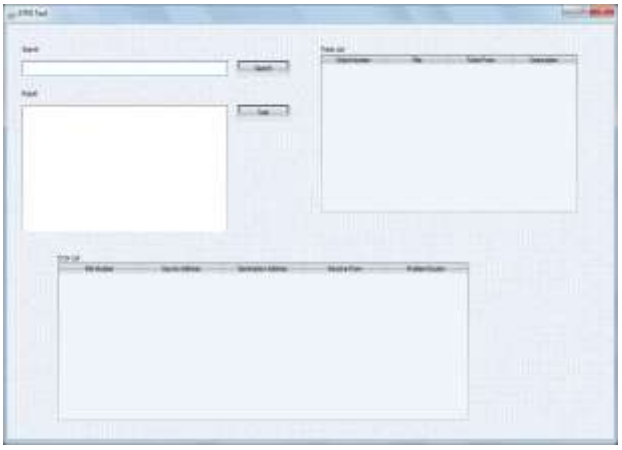
VI. EXPECTED RESULTS

The system is able to

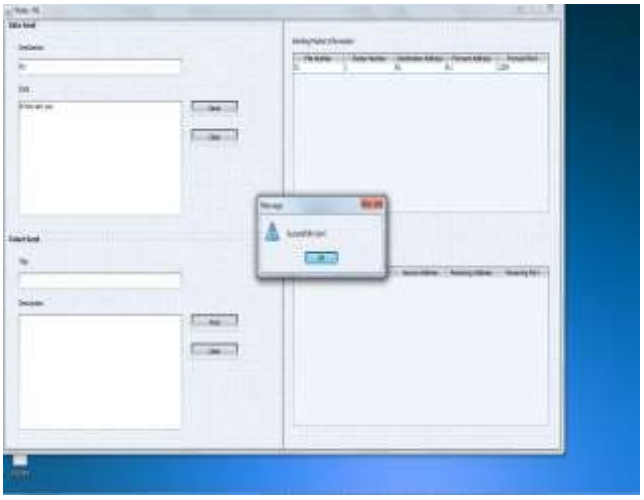
- ✓ Send test packets to all terminals
- ✓ Check liveness properties of network.
- ✓ Find faults if occurs.
- ✓ Show the forwarding state of packets

The Result of the system contains the Screen shots as bellow.

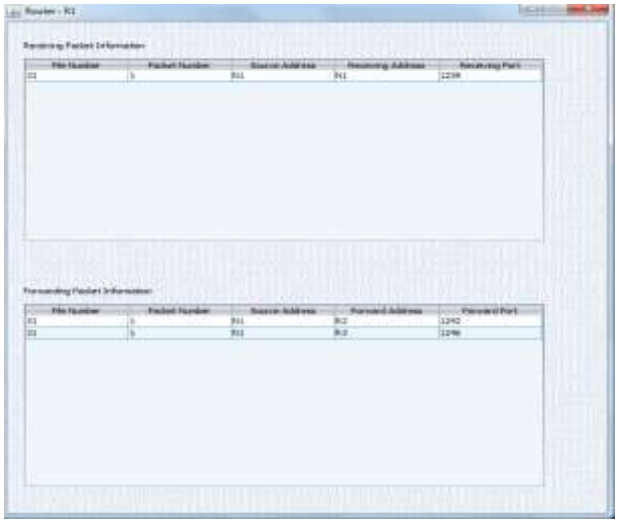
1. Test Packet Generation Tool:-



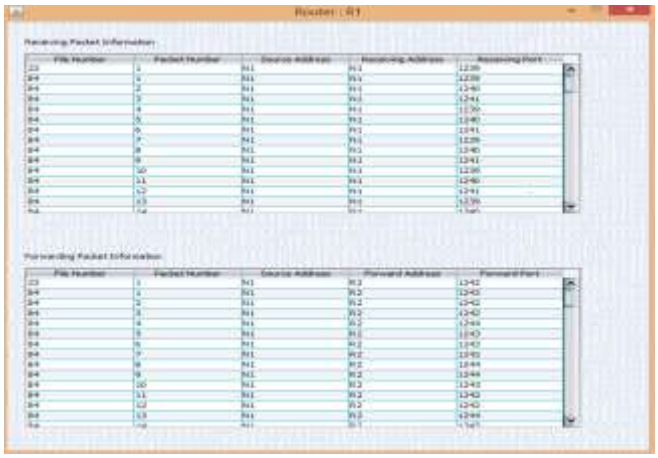
2. Node N1 sending packet information-



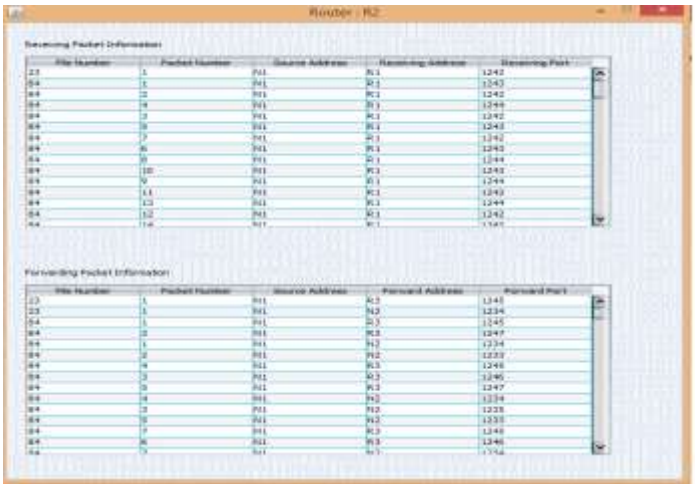
3. Receiving packet information at router R1-



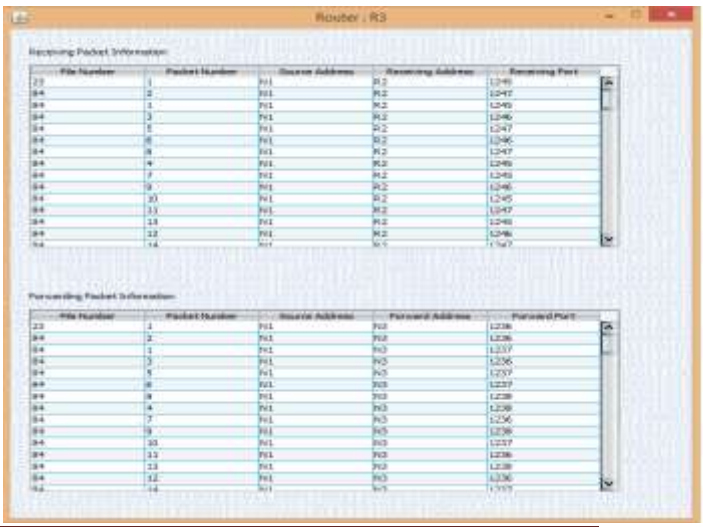
4. Admin monitoring the network by checking router R1



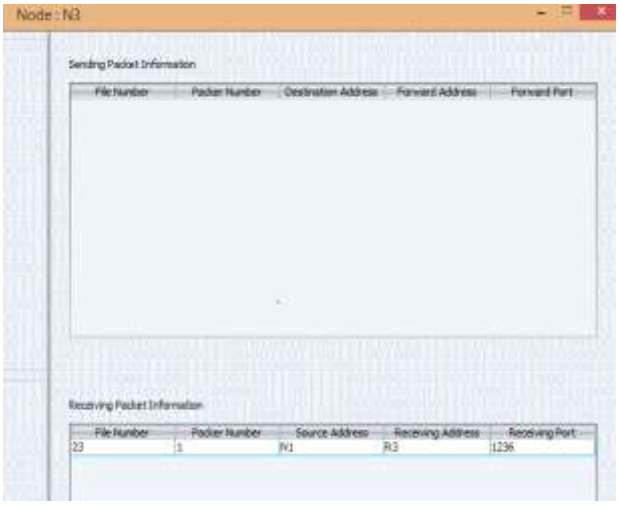
5. Admin monitoring the network by checking router R2.



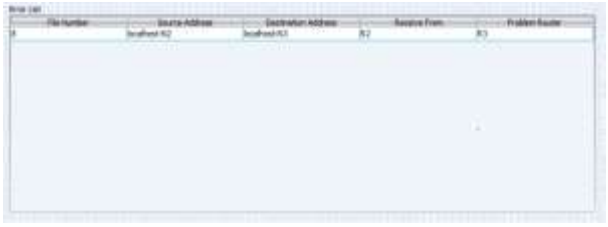
6. Admin monitoring the network by checking router R3



7. The receiving packet information at node N3



8. Tool displaying the faulty router in the error list.



9. Testing of search button

VII. CONCLUSION

Current System uses a method which is neither exhaustive nor scalable. Even though it reaches all the pairs of edge nodes it fails to detect faults in liveness properties. Automatic Troubleshooting of network using Test Packet Generation, however, goes much further than liveness testing with the same framework. Proposed system 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.

ACKNOWLEDGMENT

Firstly, I would like to express my sincere gratitude to my guide Prof.Amrit Priyadarshi for the continuous support of my project work and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this paper could not have imagined having a better advisor and mentor for my project study.

REFERENCES

[1] Hongyi Zeng, *Member, IEEE*, Peyman Kazemian, *Member, IEEE*, George Varghese, *Member, IEEE, Fellow, ACM*, and Nick McKeown, *Fellow, IEEE, ACM* IEEE/ACM "Automatic Test Packet Generation",TRANSACTIONS ON NETWORKING, VOL. 27, NO. 2, May 2015.

[2] P. Barford, N. Duffield, A. Ron, and J. Sommers, "Network perfor mance anomaly detection and localization," in *Proc. IEEE INFOCOM*, Apr. , pp. 1377–1385.

[3] 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.

[4] 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.

[5] M. Canini,D.Venzano, P. Peresini,D.Kostic, and J. Rexford, "A NICE way to test OpenFlow applications," in *Proc. NSDI*, 2012, pp. 10–10.

[6] N. Duffield, "Network tomography of binary network performance characteristics," *IEEE Trans. Inf. Theory*, vol. 52, no. 12, pp. 5373–5388, Dec. 2006.

[7] "ATPG code repository," [Online]. Available: http://eastzone.github. com/atpg/

[8] "Automatic Test Pattern Generation," 2013 [Online]. Available: http://en.wikipedia.org/wiki/Automatic_test_pattern_generation

[9] A. Dhamdhere, R. Teixeira, C. Dovrolis, and C. Diot, "Netdiagnoser:Troubleshooting network unreachabilities using end-to-end probes androuting data," in *Proc. ACM CoNEXT*, 2007, pp. 18:1–18:12.

[10] 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.

[11] "Hassel, the Header Space Library," [Online]. Available: https://bitbucket.org/peymank/hassel-public/

[12] Internet2, Ann Arbor, MI, USA, "The Internet2 observatory data collections,"[Online].Available:http://www.internet2.edu/observatory/archive/data-collections.html

[13] R. R. Kompella, J. Yates, A. Greenberg, and A. C. Snoeren, "IP faultlocalization via risk modeling," in *Proc. NSDI*, Berkeley, CA, USA,2005, vol. 2, pp. 57–70.

[14] 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.

Mr. Udaysingh Mohanrao Bhosale. Received his B.E. degree in Information Technology from University of Solapur in 2010. He has 6 years of teaching experience. He is currently working toward the M.E. Degree in Information Technology from University of Pune. His research interests lies in Networking, Network Security, Database Management System, Software Engineering and Business Process Management.

Prof. Amrit Priyadarshi received his B.E. degree in Electronics Engineering and MTech in Computer Science and Engineering. He has 10 years of experience as Assistant professor and currently pursuing PHD.