Troubleshooting a chain of networks is getting to be harder, yet organizations and authorities depend on basic instruments, for example, ping and trace route to resolve the issues. Here, the proposed system is implemented which can test and debug the network path in a computerized and standardized manner by generating test packets to act upon every connection and also on every network processing rule in the network. This approach also detects functional problems and generates packets in an automatic way to evaluate performance and sets the guidelines in processing the packet between nodes. Periodically testing packets have to be sent, if any, flaws are detected, it brings out a special mechanisms to localize the fault.

In computer networks, the proposed framework is used for automatic dynamic analysis. While transferring data, if there is any physical or software problems with the path, it leads to loss of data. With the help of the proposed system, there will be no loss of data and can minimize the cost of transferring. The proposed system makes use of Header Space Analysis framework [3] [5]. A Test packet selection algorithm is used for computing a small number of test packets which is used to test all the packet processing rules and also "Fault localization algorithm" is useful for pointing out the faulty rules and perform the end-to-end testing in the network path.
# TABLE OF CONTENTS

Abstract ......................................................................................................................... ii

Table of Contents .......................................................................................................... iii

List of Figures ................................................................................................................ v

List of Tables ................................................................................................................ vii

1. Introduction and Background ..................................................................................... 1

   1.1 Introduction ............................................................................................................... 1

       1.1.1 Network Problems ............................................................................................... 2

       1.1.2 Popular Techniques .............................................................................................. 3

       1.1.3 Survey on network issues ....................................................................................... 5

   1.2 Background .............................................................................................................. 7

2. Narrative ...................................................................................................................... 9

   2.1 Problem Statement ................................................................................................... 9

   2.2 Scope ....................................................................................................................... 9

   2.3 Motivation ............................................................................................................... 10

   2.4 Existing System ..................................................................................................... 10

   2.5 Proposed System .................................................................................................. 12

   2.6 System Architecture ............................................................................................. 19

   2.7 System Design ....................................................................................................... 21

3. Functionality of the Application .................................................................................. 22

   3.1 Module Description ............................................................................................... 22
3.1.1 Test Packet Generation ................................................................. 22
3.1.2 Generating table with all-pair reachable conditions ....................... 23
3.1.3 Localization of faults .................................................................. 25

4. Object Oriented Analysis and Design .................................................. 26

5. Evaluation and Results ..................................................................... 31
   5.1 Test case for packet flow .............................................................. 32
   5.2 Test case by incorporating fault path ............................................ 36
   5.3 Testing of “Search” button in tool ................................................. 39

6. Conclusion and Future work ............................................................... 40

Bibliography and References ................................................................. 42

Appendix ............................................................................................ 44
LIST OF FIGURES

Figure 1.1: Operational network cycle.................................................................4

Figure 1.2: Reports showing tickets and time taken to resolve tickets.................6

Figure 2.1: Creating a node..................................................................................12

Figure 2.2: Creating a router...............................................................................13

Figure 2.3: Node N1 is created .........................................................................13

Figure 2.4: Router R1 is created .......................................................................14

Figure 2.5: The tool sending test packets to all the nodes .........................15

Figure 2.6: Message is sent successfully from N1 to N3 .............................16

Figure 2.7: Packet generating tool interface .................................................18

Figure 2.8: Block diagram of the system .......................................................19

Figure 2.9: Flow chart of the system ...............................................................21

Figure 3.1: Structure of the network topology with three switches ..........23

Figure 4.1: Use case diagram for user ..........................................................26

Figure 4.2: Use case diagram for network administrator .........................27

Figure 4.3: Sequence diagram for user ..........................................................28

Figure 4.4: The activity diagram showing the step by step activity of the user ...29

Figure 4.5: Collaboration diagram depicting the interactions of the user ....30

Figure 5.1: Network topology created to implement the project ...............31

Figure 5.2: The sending packet information at node 1 ...............................32

Figure 5.3: Admin monitoring the network by checking router R1 ..........33
Figure 5.4: Admin monitoring the network by checking router R2 .................34
Figure 5.5: Admin monitoring the network by checking router R3 .................35
Figure 5.6: The receiving packet information at node N3 ..................................36
Figure 5.7: Tool displaying errors and tickets .......................................................37
Figure 5.8: Tool displaying the faulty router in the error list. ..............................38
Figure 5.9: Testing of search Button..................................................................39
LIST OF TABLES

Table 1.1. Administrators reported the symptoms and causes rankings ...............5

Table 1.2. Some of the tools used by admins in an operational network ..........7

Table 2.1. Chart showing information about functionalities of existing system ...11

Table 3.1. All-Pair reachability table ................................................................24
1. INTRODUCTION AND BACKGROUND

1.1 Introduction

Operating a current modernized network is not an easy task. Every day engineers strive hard with the links which are misconfigured, software bugs and endless other bounds that make the network work improperly or lead to network breakdown. Administrators contribute compelling effort in ensuring that the network meets their intended policies. While the recent effort on checking accuracy, reliability and meeting such type of policies have become a huge struggling task. Network operators depend on the basic tools such as ping, traceroute etc. Debugging the networks is getting harder as networks are becoming greater in number (a current server center accompanying 15,000 switches, a college grid serving 50,000 to 1 lakh users) [16] and becoming more sophisticated (which may contain more than 6,000 RFC pages, router software which can accommodate million lines of source code, billions of gates in the chips of a network) [8].

For some great reasons, troubleshooting is a burdensome work. The primary reason is due to forwarding state is distributed over different switches and firewalls and is determined by forwarding tables and other composition parameters. The second reason is, forwarding state is very difficult to examine because it typically requires manual way of logging into each and every box in the network via Command Line Interface (CLI). The third reason is, there are various programs, convention sets, rules and individuals modernizing the forwarding state concurrently which makes the debugging a complex task. Because of this issues, network engineers deserve better tools other than ping and traceroute. To address this challenge, the proposed system performs an automated process
in a standardized way to detect bugs and faults among the networks by generating test packets to act on each and every link in the network and also on every rule in the network.

1.1.1 Network Problems

There are many problems in the network. Here are the top most issues that are affecting today’s large computer networks [13].

- Performance Degradation
- Identification of Host
- Issues in security.

- Performance Degradation is related to issues like loss of speed, integrity of data due to weak communications. Complex networks are affected due to the additional partitions, endpoints, and additional equipment at midpoints.

- Configuration of proper network is key for maintaining the proper host ID. Generally, the mail station can't convey messages without proper addressing, and without proper networking equipment. While simpler networks can easily be arranged by addressing the configuration in a manual way, coming to the complex systems this way of addressing becomes totally unrealistic [11]. One can make expansive and versatile network by taking help of domain controllers, DHCP servers and their essential addressing software and protocols.

- Maintaining uprightness in the system, blocking unauthorized users from penetrating the system and ensuring the system foreswearing of administration assaults should be followed in order to maintain security.
1.1.2 Popular Techniques

For every network administrator, network troubleshooting tools are a necessity. It is very hard to detect performance problems and end-to-end fault detection in wide area networks due to increase in the complexity of paths and network dependency. To diagnose and monitor different network metrics and collect monitoring information from a vast number of hosts around the globe, there are several monitoring infrastructures. Here there is description about tools which are used to troubleshoot common networking issues [15]:

**Ping:** It is one of the common tools in networking, which informs about connections by conducting a basic test between hosts which are requesting and responding. Providing the key idea of the area that is affected in the network is the useful feature in this tool.

**Traceroute:** Once the key idea of the affected area is provided by the ping tool, traceroute is utilized in order to produce more particular information between hosts and the destination path.

**Ipconfig/ifconfig:** This tool finds out the particular IP configuration of infected hosts. If addressing is static configuration, this information is already known, but if the addressing is dynamic configuration, all hosts IP addresses can change frequently.

**Putty/Tera Term:** A Secure Shell or telnet is needed at the point when interfacing with various kinds of equipment; when all these are obliged both the Putty and Tera Term projects have the capacity to give these functionalities.

**Speedtest.net/pingtest.net:** It is used to test the quality of connection of the Internet. The bandwidth available at a particular time to a specific host can also be known by this tool.
Besides this tools, there are numerous tools ranging from basic model to complex configured systems which are useful for detecting anomalous behavior. All these different types of tools serve for specific purposes and satisfy the user needs. The admin goes through different stages in monitoring the network. Figure 1.1 shows the cycle how the operator debug the network.

![Operational network cycle](image)

**Figure 1.1 Operational network cycle [6]**

The administrator implements the network performance tool, the tools verifies every link in the network and monitors all the paths. If there are any faults in the path, it reports the administrator by different modes like alarming, signaling etc. All the reports are stored in the database, the operator can rectify the faults which cause inconvenience in the operational network.
1.1.3 Survey on network issues

A survey was conducted to the subscribers of the NANOG mailing list to comprehend the issues network engineers’ experience, and currently how they investigate them. Among the issues collected, the most common issues cannot be detected by static checks and require some form of dynamic testing tool like the proposed work. Even the most common failures in the network, failures in the reachability and violations in the policies of security also desire automatic packet generation-like dynamic testing. Table 1.1 shows the common problems faced by administrators.

**TABLE 1.1 Administrators reported the symptoms and causes rankings**

(MORE OFTEN; LESS OFTEN). In the table the column % of ≥4 represents the percentage reported greater than 4: symptoms of network failure [8]

<table>
<thead>
<tr>
<th>Category</th>
<th>Avg</th>
<th>% of ≥4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability Failure</td>
<td>3.67</td>
<td>56.90%</td>
</tr>
<tr>
<td>Throughput/Latency</td>
<td>3.39</td>
<td>52.54%</td>
</tr>
<tr>
<td>Intermittent Connectivity</td>
<td>3.38</td>
<td>53.45%</td>
</tr>
<tr>
<td>Router CPU High Utilization</td>
<td>2.87</td>
<td>31.67%</td>
</tr>
<tr>
<td>Congestion</td>
<td>2.65</td>
<td>28.07%</td>
</tr>
<tr>
<td>Security Policy Violation</td>
<td>2.33</td>
<td>17.54%</td>
</tr>
<tr>
<td>Forwarding Loop</td>
<td>1.89</td>
<td>10.71%</td>
</tr>
<tr>
<td>Broadcast/Multicast Storm</td>
<td>1.83</td>
<td>9.62%</td>
</tr>
</tbody>
</table>

Referring to the Figure 1.2, every month more than 100 tickets are from the 35% of the systems. To solve an issue, it takes nearly 30 minutes and was reported by 40.4% of
respondents. It holds averagely 60 minutes time was reported by 25% of respondents [10].

![Pie charts showing ticket distribution and time taken to resolve tickets.](image)

**Figure 1.2.** Reports showing the (a) tickets issued monthly in the network and (b) time taken to rectify the tickets. [8]

Causes: Dynamic testing is used to find out the common concerns such as router and software bugs.

Cost in Troubleshooting: There are two measurements that can figure out the amount of network debugging - number of tickets that are generated in the network related issues at regular intervals and time expended to solve the issue.

Tools: From table 1.2, ping, traceroute and SNMP are the very popular tools used by administrators. But in the survey, one can see that around 70% of the people are in the need of tools which can detect automatically by sending test packets.
Table 1.2. Some of the tools used by administrators in an operational network

(More often=5; Less Often=1) [8]

<table>
<thead>
<tr>
<th>Category</th>
<th>Avg</th>
<th>% of ≥ 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ping</td>
<td>4.50</td>
<td>86.67%</td>
</tr>
<tr>
<td>traceroute</td>
<td>4.18</td>
<td>80.00%</td>
</tr>
<tr>
<td>SNMP</td>
<td>3.83</td>
<td>60.10%</td>
</tr>
<tr>
<td>Configuration Version Control</td>
<td>2.96</td>
<td>37.50%</td>
</tr>
<tr>
<td>netperf/iperf</td>
<td>2.35</td>
<td>17.31%</td>
</tr>
<tr>
<td>sFlow/NetFlow</td>
<td>2.60</td>
<td>26.92%</td>
</tr>
</tbody>
</table>

1.2 Background

In a network, information is exchanged in the form of packets to reach their destination. A packet contains header and payload. The header part contains all the information regarding source, destination addresses, rules, etc. The payload is the actual data which is to be communicated to the nodes. At different levels several researchers have worked on these parameters. Ubik, Anoniades and Osbelo [14] utilized a packet capturing system to screen the activity going through the router connected with the organization server. Their system lives up to expectations by consistently observing the inbound and outbound activity through a router without a predefined data transfer capacity designation to a specific host or client. They additionally proposed the transmission capacity distribution on an OSI model. This technique helps in analyzing the overall transmission capacity usage in the seven layered architecture without the requirement for data about the information stream prerequisites.
One does not have a clear understanding of the previous strategies which can generate the “test packets” automatically from the network configurations [9] [13]. In large networks, scalability is the main challenge to be dealt with since the data packet transferring leads to data loss due to the complex infrastructure. To address this, “NICE” [2] model can be used to augment the model checking with the symbolic execution of event handlers. It contains effective methods for producing event inter-leavings likely to uncover bugs. The “Header Space Analysis” [3] framework is the basis for the proposed model. The way in which packets are processed is shown in the “header space framework” as a geometric model [3] [5].

Checking the network and diagnosing the faults in the network End-to-End path in static way has been used in the previous works [4]. By contrast, the developed system can check end-to-end paths by exercising every rule and every link in the network. By making use of router configurations and data plane information, the developed model improves the detection granularity to the rule level. Therefore, this approach not only checks network-liveliness but also implements high level properties such as performance.

At present, there are so many approaches which are based on measurement friendly-environment. They are correlative to the proposed approach, by integrating input and output requirements it can produce automatic test packets. The work in symbolic debugging techniques and the proposed work are somewhat similar. The proposed methodology is faster [8] since one can calculate the packet’s forward path directly instead of solving the constraints, whereas KLEE [1] is found 15 times backwards when compared to header space framework [8] [7].
2. NARRATIVE

2.1 Problem Statement

Debugging the network automatically in an efficient manner is a challenging task using network monitoring tools. Working in static manner is the feature which can be seen in existing network debugging tools [2] [3] [9] [14]. These methodologies are not intended to diagnose liveliness failures occurred by bugs in the software, damaged routers, bugs due to frizzled connections and switches, or performance problems generated by congestion in the network. Researchers have proposed tools which enforce consistency between devise configuration and policy.

2.2 Scope

The two main reasons for failures in network are software bugs and hardware problems, and that issues shows themselves both as deterioration in throughput and reachability failures. Automatic detection of these types of faults is the objective of this research.

The project scope is described as follows:

1) Monitor the network by sending testing packets to all the nodes present in the network.

2) Displays the messages regarding success and failure of the information delivered from the source to the destination.

3) If the users raise any issues regarding the network, the admin can observe the tickets raised by the users and can rectify the particular faulty device.
With the help of this tool, one can see the ports available by sending test packets. Then the user can send the original message. If the path is obstructive, it leads to the loss of data, thereby, increase in the cost of communication.

2.3 Motivation

The complex nature of modern networking system, the hierarchy among the components and security levels in the system become more sophisticated, which make the administrator difficult in understanding the complex interactions between components behavior. This lead to the difficulties in the prediction and control of faults in the network. Packet forwarding in advanced systems is a complex methodology, including mutually dependent functions running on large number of devices.

The Existing tools are inadequate since they take timescales of seconds to hours, since they operate offline and check network configuration files and the data-plane state. The concept of evolving an automated test packet generation device prompts inspiration to work towards developing up this project.

2.4 Existing System

In most of the cases in the existing works, the admin used to manually decide which ping packets to send. Here, the current system methodologies designed can avert errors in software logic, however, neglect to detect the errors caused by fizzled connections and switches. The Table 2.1 provides some information about the functionalities of existing works and their behaviors [12].
Table 2.1 Chart showing information about functionalities of the existing system [12]

<table>
<thead>
<tr>
<th>Parameter Reference</th>
<th>Fault Propagation</th>
<th>Fault Activation</th>
<th>Hardware Failure</th>
<th>Software Bug</th>
<th>QOS (B/W, Latency, Throughput)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Test Packet Generation</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>PlanetLab All-ping Service</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Robust monitoring faults and delays</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Network tomography of binary network</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

Working in a static manner is the feature which can be seen in existing network debugging tools [2] [3] [9] [10]. These methodologies are not intended to diagnose liveliness failures affected due to damaged router architecture, or problems generated by
congestion in network operational performance. With these approaches, one can prevent logical errors occurred in control plane software. Researchers have proposed tools which enforce consistency between devise configuration and policy.

### 2.5 Proposed System

The system generates test packets to act upon all the links in the network that covers all the forwarding rules covered and exercised by a minimum of one packet to test.

In the first phase, creation of nodes and routers takes place. Here in this project, the network is virtualized without using any open source network simulators. Virtual nodes and router interfaces are created with the help of system design. Three nodes and three routers are created. Each node has a corresponding router like node ‘N1’ has router ‘R1’ router and ‘N2’ has corresponding ‘R2’ and similarly ‘N3’ have ‘R3’. The number of nodes and routers are predefined in this project to three in number. One can send messages from single node to another node via router.

The nodes and routers are created by entering the node number and router number in the form fields. The creation of the node and router interface is shown in figures 2.1 and 2.2.

![Figure 2.1. Creating a node](image)
Figure 2.2. Creating a router

After entering the particular node number and router number, a node and router are created which is shown in the figures 2.3 and 2.4, Likewise, the rest of the nodes and routers are created.

Figure 2.3. Node N1 is created
Figure 2.4. Router R1 is created.

Initially, the packet generating tool generates the test packets. By doing so, it displays an alert saying test packets are sent to all the nodes which is shown in figure 2.5.
Figure 2.5. The tool sending test packets to all the nodes

After that, a message can be sent from one node to another node through the routers. The destination node and the data should be specified in the space available in the interface of the source node. After packet reaching a desired destination, then it will pop an alert message displaying “Successfully Sent” which is shown in figure 2.6.
Figure 2.6. Message is sent successfully from N1 to N3

All the information regarding the source address, forward address, file number, packet number, receiving port, forwarding port can be seen in the router interface. If a message is sent from (N1) to (N2), the N1 interface displays “Sending Packet Information” which has file number, packet number, destination address, forward address, and forward port (R1). Now admin checks the router interface of (R1) for “Forwarding Packet Information” which has file number, packet number, destination address, forwarding address and forwarding port (R2). Next the admin checks the “Receiving Port Information” of router (R2) which displays the Receiving Address as R1 which means the data has come from the N1.
In another phase of the project, monitoring the faults by using the “Fault Localization Algorithm” is performed. Among the three nodes, one node should be disabled. Then a message should be sent in between the remaining nodes. In the tool interface, there are some slots. They are-

File number: the affected file is displayed

Source address: This contains the information from where the packet is sent

Destination address: To which location the packets are transferring.

Receive from: This slot has the router information from where the ticket has raised.

Problem Router: This contains the node number which is affected.

In the third phase of the project, a user-friendly environment is developed. If a user feels that any particular node is not working properly, one can raise tickets to the administrator of the network. The user can enter the details about the faulty nodes in the “Ticket list” section present in the Node’s Interface. The details can be posted by filling the “title of the issue” and “description” slots. The interface of the tool has two sections “Error list” and “Ticket list” from where the admin can observe the issues. The Ticket list section contains the information about user raised tickets. The section is divided into some parts. They are:

Ticket Number: The ID of the particular ticket

Title: The title of the ticket filled by the user is displayed in this slot.

Ticket From: From which node the ticket is raised.
Description: The complete description of the faults can be seen in this area.

The interface of the packet generating tool is shown in figure 2.7:

Figure 2.7. Packet generating tool interface
2.6 System Architecture

The overall architecture of the proposed work is displayed in Figure 2.8.

![Figure 2.8. Block diagram of the system [8]](image-url)
The detailed description about the block diagram is represented in the form of steps:

Step 1: The framework first gathers all forwarding states in the network path across the various nodes and routers present in different locations. The information in the collected forwarding states contains configuration files, topology of the network, access control lists, forwarding information base tables (FIB’s).

Step 2: Header Space Analysis framework [3] is used in order to figure out the reachability level in between all the end-to-end terminals.

Step 3: “Test Packet Selection Algorithm” uses the computed outcome in the step 2 to select test packets to examine all the packet processing rules.

Step 4: The test terminals will send the test packets from time to time in an orderly fashion.

Step 5: The “Fault localization algorithm” will come into role whenever an error took place by reducing the error effects during the above process.

The steps from 1 to 3 are taken from [3], step 4 and 5 are implemented in the proposed system.
2.7 System Design

Initially, the user send the data from one node to another via routers. If the path is clear, the user can get the reply. If the path is faulty, the packets are redirected to packet generating tool. The tool checks the failures and clears the path and notifies the admin about the faulty nodes and faulty routers.

Figure 2.9. Flow chart of the system
3. FUNCTIONALITY OF THE APPLICATION

3.1 Module Description

Following, there is a description of the modules implemented in this project:

1. Test Packet Generation
   a. Test Packet Selection

2. Generating Table with All-Pair Reachable conditions
   a. Sampling
   b. Compression

3. Localization of faults

3.1.1. Test Packet Generation

Let us assume that the sending and receiving of test packets is occurring in between a set of test terminals. Generating minimum number of “test packets” to execute all the packet processing rules in every node is the main aim of the proposed work. The system must respect two key constraints, while generating the test packets. 1) Port - The tool should only make use of available test terminals, 2) Header – The tool uses the headers for the purpose of granting send permission to every testing terminal.

*Test Packet Selection:* The switching functions \{T_1, \ldots, T_n\}, and network topology function, \(\hat{T}\), the Test Packet Selection (TPS) [8] [7] algorithm can be used for selecting the test packets to exercise all packet processing rules which can be reachable, subject to the header and port constraints in a network. In between each
and every pair of available nodes, Test Packet Selection algorithm (TPS) is used for finding the equivalent set of classes.

### 3.1.2. Generating table with all-pair reachable conditions

While the packet is transferred in between one test terminal to another test terminal, the tool starts working by calculating the total set of packet headers in the first step. Then it computes an entire rule sets so that it exercises along the path from each such packet header. It takes use of all-pair reachability algorithm described in [3] in order to find the possible pairs. The header with all wild carded bits i.e., all-x header is combined with the transfer function \( \Gamma \) connected to each test terminal. Using the network function, the packet \( g_k \) moves across the network, the rules set which matches with the \( g_k \) are recorded in the \( g_k \) history. By performing this type of matches for every test terminal pair, the all-pair reachable table is created which is depicted in the Figure 3.1.

![Figure 3.1 Structure of the network topology with three switches](image)

**Figure 3.1 Structure of the network topology with three switches [8]**
For example, test packet would go through the switch 1 which is inserted in Node N1. The forwarded packets from switch 1 will go to switch 2 with dst_ip=10.0/16 and then with dst_ip=10.1/16 go to switch 3. Table 3.1 is the respective all-pair reachable table.

**Table 3.1. All Pair reachability table [8]**

<table>
<thead>
<tr>
<th></th>
<th>Header</th>
<th>Input Port</th>
<th>Output Port</th>
<th>Rules History</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>dst_ip=10.0/16, tcp=81</td>
<td>N1</td>
<td>N2</td>
<td>link N1N2</td>
</tr>
<tr>
<td>p2</td>
<td>dst_ip=10.1/16</td>
<td>N1</td>
<td>N3</td>
<td>link N1N3</td>
</tr>
<tr>
<td>p3</td>
<td>dst_ip=10.2/16</td>
<td>N2</td>
<td>N1</td>
<td>link N1N2</td>
</tr>
<tr>
<td>p4</td>
<td>dst_ip=10.1/16</td>
<td>N2</td>
<td>N3</td>
<td>link N2N3</td>
</tr>
<tr>
<td>p5</td>
<td>dst_ip=10.2/16</td>
<td>N3</td>
<td>N1</td>
<td>link N2N3</td>
</tr>
<tr>
<td>(p6)</td>
<td>dst_ip=10.0/16, tcp=80</td>
<td>N3</td>
<td>N2</td>
<td>link N2</td>
</tr>
</tbody>
</table>

The Header column describes destination IP addresses including port numbers. The ‘Input Node’ column describes the source node and ‘Output Node’ column describes destination node. The rule history column specifies the link in which packet travels from source to destination.

**Sampling:** In the next step, the tool selects minimum of a test packet to act upon all the reachability rules. Selecting a single packet per class in some random manner is the simplest mechanism.

**Compression:** In this step, the packets are compressed to minimum number. So, the system selects a minimum set of packets from the sampling step, such that it covers all the rules of the union of rule histories. The cover is selected in such a way that all connections
or all router configurations are covered. This is known as Min-Set-Cover problem [8]. While NP-Hard, greedy $O(N^2)$ gives a decent approximation, here $N$ represents the test packets. The obtained minimum set of test packets is referred as regular test packets $\{p_1, p_2, p_3, p_4, p_5\}$ and the remaining packets which are not selected comes under reserved test packets $\{p_6\}$.

3.1.3 Localization of faults

After sending the minimum set of test packets across the node, it tests for failures. If any faults are present in the path, the system diagnosis the defects.

The Result function $R$ contain the data about failed rules. The Result function is defined for rule $r$ are as follows:

$$ R(r, gk) = \begin{cases} 
0, & \text{if } gk \text{ has faults at rule } r \\
1, & \text{if } gk \text{ has no faults i.e succeeding at rule} 
\end{cases} $$

Forwarding rule$ = $ checks whether the test packet is delivered at the destination node or not

Link Failure$ = $ If the forwarding rule fails in the topology, it causes link failure.
4. OBJECT ORIENTED ANALYSIS AND DESIGN

During the software development process, the project is followed by Unified Modeling language (UML) rules in drawing UML diagrams. The operation of processes with each other and the order of operation is shown in sequence diagrams. The graphical overview of the system functionality is shown in use case diagrams.

The Use case Diagram for the user is shown in the figure 4.1. In this diagram, it represents the role of the user working with the tool. The user sends the data to another node via router which is connected to the tool. The tool generates the packets and forwards to all the nodes. The user can raise the tickets, if there are any network problems.

USE CASE DIAGRAMS

![Use Case Diagram for User](image)

Figure 4.1.Use case diagram for user
Figure 4.2 shows the use case diagram for the Network Administrator. The tool will be accessed by the network admin. The packet generation is the feature present in the tool. The admin searches the network for any problems. If there is a problem, then the admin calls for the tool which checks the network for any faults.

![Use case diagram for network administrator](image)

**Figure 4.2. Use case diagram for network administrator**

The sequence diagram is shown in the figure 4.3. Initially the user sends the information to the router. From there, the router forwards the information. If there is any failure takes place, it invokes the Test packet Generation tool. This tool generates the packets that tests the network and show the message if the packet is not delivered. If the path is clear the user can receive the data.
Figure 4.3. Sequence diagram for user

Figure 4.4 shows the activity diagram. This diagram shows the stepwise activities done by the user with the help of all the components of the system. The overall control flow of the elements in the system is shown in the activity diagram from the start state to final state. The user send the data via router. The Test Packet Generation tool generates the
test packets and passes to routers and nodes present in the network. If the path has no faults then the data is send to another user. Upon failure, the tool displays the area where the problem occurred.

**ACTIVITY DIAGRAM:**

![Activity Diagram](image)

Figure 4.4. The activity diagram showing the step by step activity of the user
The Collaboration diagram is shown in figure 4.5. The user sends the data to another user via router. Then the router forwards the information to forwarding router which is in the network path to reach the destination. Upon success, the user can view the data. Upon failure, it sends the problem to the developed tool, the tool checks the problem by generating the test packets which displays the network faults in the error list. The users can even raise the tickets to the admin by posting. The admin checks the user tickets in the tool interface.

**COLLABORATION DIAGRAM:**

![Collaboration Diagram](image)

*Figure 4.5. Collaboration diagram depicting the interactions of the user*
5. EVALUATION AND RESULTS

In this section, the developed project functionality is judged by performing different cases of testing and thereby evaluating. Initially, the flow of the packet across the routers to reach a particular is tested. The node module has “Sending Packet Information” and “Receiving packet Information” slots which can be seen in the interface. This two slots provide the information about the packet flow. The Router module has “Receiving packet Information” and “Forwarding packet Information” sections which provide the information about the location, port number etc., of the packet.

Figure 5.1 shows the network topology used in the project implementation. Here each node is connected to router and the information is passed to destination node in the form of packets.

Figure 5.1 Network topology created to implement the project
5.1. **Test case for Packet flow:** Here the information is sent from node 1 to node 3. In figure 5.2, the ‘Forward Address’ slot, R1 represents the packet has transferred to the router 1. The ‘Forward Port’ represents the port number to which the packet should be sent.

![Sending Packet Information](image)

**Figure 5.2. The sending packet information at node 1**

Now, the admin checks the R1 module’s “Receiving Packet Information” and “Forwarding Packet Information” divisions which tells about the packet’s further flow in reaching the destination to Node 3. In figure 5.3., the source address represents packet origin, forward address represents to which location the packets are transferring, file number represents the packet ID, packet number, receiving port represents the port number from where the packet is receiving and forwarding port represents the port number where the packet should be sent.
Figure 5.3. Admin monitoring the network by checking router R1

The router has packet information about the original message and also it includes the information about the test packets. The test packets are broadcasted in order to exercise each and every link. That is why there are so many entries in the router interface.
In R1’s Forwarding Packet Information, the packet is transferring to R2 router. Now the admin opens the R2 module interface shown in the figure 5.4.

![Figure 5.4 Admin monitoring the network by checking router R2](image)

In R2 forwarding packet information, it is shown that packet is received from R1 and then it is located at the node 2 from there it is forwarded to R3. Now, the admin checks the R3
module which is shown in figure 5.5. Every time the admin have to check the flow by observing the file number.

Figure 5.5. Admin monitoring the network by checking router R3
Finally the packet is forwarded to node 3 which is the final destination. It can be seen from the Forward Address= N3 in R3 module’s Forwarding Packet Information.

At Node 3, all the information regarding the packet journey can be seen which is shown in figure 5.6.

![Node N3](image)

**Figure 5.6. The receiving packet information at node N3**

**5.2. Test Case by incorporating fault path:** The faulty path is set by removing either node or router.
By removing the node: In this case, disable the Node 2 and send the message from Node 1 to Node 3. The information regarding the faulty node will be displayed in the Packet Generating tool in the Error List. The tickets raised by the users regarding the network problems will be displayed in the Ticket List. This can be shown in the figure 5.7.

Figure 5.7. Tool displaying errors and tickets in the form of lists
By removing the router: In this case, disable the Router 3 and send the message from Node 2 to Node 3. The information regarding the faulty router will be displayed in the Packet Generating tool in the Error List. This can be shown in figure 5.8.

![Error List](image)

**Figure 5.8.** Tool displaying the faulty router in the error list.
5.3 Testing of “Search” button in the Tool: If the admin wants to see the list of problems at particular node or router, then the admin clicks on the “search” button in the Tool interface which is shown in the figure 5.9

![Figure 5.9: Testing of search button](image)

Figure 5.9: Testing of search button
6. CONCLUSION AND FUTURE WORK

Today's systems require an abundant human interaction to keep them working. As systems get complex there is colossal enthusiasm for robotizing the control, error reporting, investigating and trouble shooting. For the operators with huge information centers and Inter Service Providers, testing the network health is a very cumbersome task, since the ISP’s should offer their customers more services rapidly without any interruption. The developed system can test the liveliness of the network by automatically generating the minimum set of test packets [8] (Min-Set-Cover by using union of rule histories [17]). It reads information about forwarding tables from all the routers and builds up a device independent model based on Header-Space Analysis.

Automatic Test Packet Generation type tools are in dire need in checking liveliness of the network. The developed tool works in dynamic way of checking liveliness by testing all the rules, thereby achieving the reachable policies. In case of any faults during the information exchange, a fault localization algorithm is used for detecting the failed links across the nodes in the system.

The execution likewise expanded testing with a straightforward fault localization algorithm which is additionally developed by utilizing the header space structure. As in programming testing, the formal model assisted to increase the testing analysis there by decreasing the number of test packets. The outcome generated displays that every link in the network can be operated with limited amount of packets [8]. One can trust the developed project will be just as helpful for robotized element testing of complex systems.
since it can minimize the data loss by sending the original message after checking the network links and devices with test packets.

**Future Work:**

- Checking the network repeatedly in order to find the forwarding state is main reason for overhead in the developed tool. In the future, some techniques have to be developed for minimizing the overhead by developing an interface which can provide all information in a single window.
- The developed project is just a prototype by creating few nodes and routers. In the near future one can enhance by creating a large number of routers and nodes.
- The real time versions of automatic packet generation tools should be developed which can induce the latest approaches in reducing overhead.
BIBLIOGRAPHY AND REFERENCES


Appendix

Class Diagram:
Network Configuration:

*config.properties*

```plaintext
# Node
node=N1, N2, N3

# Router
router=R1, R2, R3

# Neighbour
NeighN1=R1
NeighN2=R2
NeighN3=R3
NeighR1=R2, N1
NeighR2=R1, R3, N2
NeighR3=R2, N3

# Address
IPN1=localhost
IPN2=localhost
IPN3=localhost
IPR1=localhost
IPR2=localhost
IPR3=localhost

# Port
PortN1=1230, 1231, 1232
PortN2=1233, 1234, 1235
PortN3=1236, 1237, 1238
PortR1=1239, 1240, 1241
PortR2=1242, 1243, 1244
PortR3=1245, 1246, 1247

# Tool Details
ToolIP=localhost
ToolPort=1221
TestSource=N1
TestDestination=N3
```
AtpgUtil.java:

```java
package pro.atpg.utility;
import java.io.File;
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Properties;
public class AtpgUtil {
    public static String TOOL_IP;
    public static int TOOL_PORT;
    public static String testSourceName;
    public static String testDestinationName;
    public static HashMap<String, String> ipMap;
    public static HashMap<String, ArrayList<String>> neighMap;
    public static HashMap<String, ArrayList<Integer>> portMap;
    public static String testContent = "ABCD EFGH IJKL MNOPQRST UVWX YZ";
    public AtpgUtil() {
        int total = 100;
        for (int i = 0; i < total; i++) {
            testContent += "::
        }
        ipMap = new HashMap<String, String>();
        neighMap = new HashMap<String, ArrayList<String>>();
        portMap = new HashMap<String, ArrayList<Integer>>();
        try {
            Properties prop = new Properties();
            InputStream input = new FileInputStream("E:\Software\APTG\src\config.properties");
            prop.load(input);
            String routers[] = prop.getProperty("router").split(",");
            String nodes[] = prop.getProperty("node").split(",");
            for (String router : routers) {
                ipMap.put(router, prop.getProperty("IP" + router));
            }
            ArrayList<String> tempNeigh = new ArrayList<String>();
            if (prop.getProperty("Neigh" + router).indexOf(",") != -1) {
                for (String nei : prop.getProperty("Neigh" + router).split(",")) {
                    tempNeigh.add(nei);
                }
                neighMap.put(router, tempNeigh);
            } else {
                tempNeigh.add(prop.getProperty("Neigh" + router));
                neighMap.put(router, tempNeigh);
            }
            ArrayList<Integer> tempPort = new ArrayList<Integer>();
            if (prop.getProperty("Port" + router).indexOf(",") != -1) {
                for (String port : prop.getProperty("Port" + router).split(",")) {
                    tempPort.add(Integer.parseInt(port));
                }
                portMap.put(router, tempPort);
            } else {
                tempPort.add(Integer.parseInt(prop.getProperty("Port" + router)));
                portMap.put(router, tempPort);
            }
        }
    }
}
```
for(String node:nodes){
    ipMap.put(node,prop.getProperty("IP"+node));

    ArrayList<String> tempNeigh=new ArrayList<String>();
    if(prop.getProperty("Neigh"+node).indexOf(",")!=-1){
        for(String nei:prop.getProperty("Neigh"+node).split(",")){
            tempNeigh.add(nei);
        }
        neighMap.put(node, tempNeigh);
    }else{
        tempNeigh.add(prop.getProperty("Neigh"+node));
        neighMap.put(node, tempNeigh);
    }

    ArrayList<Integer> tempPort=new ArrayList<Integer>();
    if(prop.getProperty("Port"+node).indexOf(",")!=-1){
        for(String port:prop.getProperty("Port"+node).split(",")){
            tempPort.add(Integer.parseInt(port));
        }
        portMap.put(node, tempPort);
    }else{
        tempPort.add(Integer.parseInt(prop.getProperty("Port"+node)));
        portMap.put(node, tempPort);
    }

    TOOL_IP=prop.getProperty("ToolIP");
    TOOL_PORT=Integer.parseInt(prop.getProperty("ToolPort");)
    testDestinationName=prop.getProperty("TestDestination");
    testSourceName=prop.getProperty("TestSource");
    }catch (Exception e) {
    e.printStackTrace();
    }

    public static void main(String args[]){
        File file=new File("config.properties");
        System.out.println(file.getAbsolutePath());
        new ATPUtil();
    }
Node.java:

```java
public void mainMethod(){
    new ATPGUtil();
    while(true){
        String inp=JOptionPane.showInputDialog("Enter the Node Name");
        if(inp!=null & inp.length()>0){
            nodeName=inp.toUpperCase();
            break;
        }
    }
    for(int port:ATPGUtil.portMap.get(nodeName)){
        new NodeReceiver(port,nodeName);
    }
}

public static void main(String args[]){
    Node node=new Node();
    node.mainMethod();
}
```

```java
@override
public void actionPerformed(ActionEvent e) {
    // TODO Auto-generated method stub
    Object actione=getSource();
    if(actione==sendButton){
        String destination=destinationText.getText();
        String content=sourceText.getText();
        ArrayList<String> neigh=ATPGUtil.neighMap.get(nodeName);
        for(String str:neigh){
            Packet pac=new Packet();
            pac.sourceAddress=ATPGUtil.ipMap.get(nodeName);
            pac.sourceName=nodeName;
            pac.fromAddress=ATPGUtil.ipMap.get(nodeName);
            pac.fromName=nodeName;
            pac.toAddress=ATPGUtil.ipMap.get(str);
            pac.toName=str;
            pac.toPort=ATPGUtil.portMap.get(str).get(portSelection-1);
            pac.destinationAddress=ATPGUtil.ipMap.get(destination);
            pac.destinationName=destination;
            pac.fileName=new Random().nextInt(100);
            pac.packetNumber=1;
            pac.packetData=content;
            pac.type="original";
            pac.fileSize=content.length();
            pac.toolPocketSize=1;
            pac.packets=now ArrayList<String>();
            pac.paths.add(nodeName);
            pac.paths.add(nodeName);
```
dataText.setText(""");
}
else if(action==ticketSendButton){
    String title=ticketTitleText.getText();
    String desc=ticketDescText.getText();
    Ticket ticket=new Ticket();
    ticket.sourceName=nodeName;
    ticket.ticketNumber=new Random().nextInt(100);
    ticket.title=title;
    ticket.description=desc;
    ticket.toAddress=ATPUtil.TOOl_IP;
    ticket.toPort=ATPUtil.TOOl_PORT;
    sender.sendTicketObject(ticket);
}

Nodereceiver.java:

public NodeReceiver(int port,String name) {
    this.port=port;
    nodeName=name;
    contentMap=new HashMap<Integer, HashMap<Integer,Packet>>();
    resultList=new ArrayList<String>();
    try{
        ss=new ServerSocket(port);
        Thread thread=new Thread(this);
        thread.start();
    }catch (Exception e){
        e.printStackTrace();
    }
}

public void run(){
    //System.out.println("Start");
    try{
        while(true){
            //System.out.println("Start");
            s=ss.accept();
            ois=new ObjectInputStream(s.getInputStream());
            Object obj=ois.readObject();
            if(obj instanceof Packet){
                Packet packet=(Packet) obj;
                if(packet.type.equalsIgnoreCase("original")){
                    sender.sendSuccess(packet.sourceAddress,ATPUtil.portMap.get(packet.sourceName).get(1),"*"+packet.fileNumber);
                    Vector<String> vec=new Vector<String>();
                    vec.add("*"+packet.fileNumber);
                    vec.add("*"+packet.packetNumber);
                    vec.add(packet.sourceName);
                    vec.add(packet.fromName);
                    vec.add("*"+port);
                    if(nodeName.equalsIgnoreCase(packet.destinationName)){
                        Node.receiveInfoModel.addRow(vec);
                    }
                }
            }
        }
    }
}

else if(packet.type.equalsIgnoreCase("test")){
}
if(contentMap.containsKey(packet.fileNumber)) {
    HashMap<Integer, Packet> pacMap=contentMap.get(packet.fileNumber);
    if(pacMap.containsKey(packet.packetNumber)){
        pacMap.put(packet.packetNumber, packet);
        contentMap.put(packet.fileNumber, pacMap);
    }
}

//fully received
if(pacMap.size()==packet.toolPacketSize){
    sender.sendSuccess(APGUUtil.TOOL_IP, APGUUtil.TOOL_PORT, "-\:\:\:fileNumber");
}

} else {
    HashMap<Integer, Packet> pacMap=new HashMap<Integer, Packet>();
    pacMap.put(packet.packetNumber, packet);
    contentMap.put(packet.fileNumber, pacMap);
}

} else if(obj instanceof String){
    if(resultList.contains(obj)){
        //System.out.println(""+obj.toString());
        resultList.add(""+obj);
        JOptionPane.showMessageDialog(null, "Packet Successfully Sent");
    }
}
} catch (Exception e) {
    e.printStackTrace();
}

Router.java:

public void mainMethod(){

    new APGUUtil();
    while(true){
        String inp=JOptionPane.showInputDialog("Enter the Router Name");
        if(inp!=null & inp.length()>0){
            routerName=inp.toUpperCase();
            break;
        }
    }

    for(int port:APGUUtil.portMap.get(routerName)){
        new RouterReceiver(port,routerName);
    } public static void main(String args[]){
        Router router=new Router();
        router.mainMethod();
    }
}
Routerreceiver.java:

```java
public RouterReceiver(int port, String name) {
    routerName = name;
    this.port = port;
    try{
        ss = new ServerSocket(port);
        Thread thread = new Thread(this);
        thread.start();
    }catch (Exception e) {
        e.printStackTrace();
    }
}

public void run(){
    try{
        while(true){
            //System.out.println("Start");
            s = ss.accept();
            ois = new ObjectInputStream(s.getInputStream());
            Object obj = ois.readObject();

            if (obj instanceof Packet){
                Packet packet = (Packet) obj;
                Vector<String> vec = new Vector<String>();
                vec.add("+packet.fileNumber");
                vec.add("+packet.packetNumber");
                vec.add(packet.sourceName);
                vec.add(packet.fromName);
                vec.add("+port");
                Router.receiveInfoModel.addRow(vec);

                int packetType = packet.getType();
                if (packet.type.equalsIgnoreCase("original")){
                    ArrayList<String> neigh = ATPGUtil.neighMap.get(routerName);
                    for(String str: neigh){
                        if (!packet.paths.contains(str)){
                            Packet pac = packet;
                            pac.fromAddress = ATPGUtil.ipMap.get(routerName);
                            pac.toAddress = ATPGUtil.ipMap.get(str);
                            pac.toPort = ATPGUtil.portMap.get(str).get(portSelection - 1);
                            pac.toName = str;
                            pac.fromName = routerName;
                            pac.paths.add(routerName);

                            Vector<String> vec1 = new Vector<String>();
                            vec1.add("+packet.fileNumber");
                            vec1.add("+packet.packetNumber");
                            vec1.add(packet.sourceName);
                            vec1.add(packet.toName);
                            vec1.add("+packet.toPort");
                            Router.forwardInfoModel.addRow(vec1);
                            sender.sendPacketObject(pac);
                            if (portSelection >= 3){
                                portSelection = 1;
                            }else{
                                portSelection = portSelection - 1;
                            }
                        }
                    }
                }else if (packet.type.equalsIgnoreCase("test")){
                    ArrayList<String> neigh = ATPGUtil.neighMap.get(routerName);
                    for(String str: neigh){
                        if (!packet.paths.contains(str)){
                            // Additional code for test packets
                        }
                    }
                }
            }
        }
    }
}
```
Tool.java:

```java
public void mainMethod(){
    new ATPGUtil();
    new ToolReceiver(ATPGUtil.TOOL_PORT);
    public static void main(String args[]){
        Tool tool=new Tool();
        tool.mainMethod();
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
        Object obj=e.getSource();
        if(obj==testButton){
            String nodeName=ATPGUtil.testSourceName;
            int portSelection=1;
            int fileNumber=new Random().nextInt(100);
            for(int i=0;i<100;i++){
                ArrayList<String> neigh=ATPGUtil.neighMap.get(nodeName);
                for(String str:neigh){
                    Packet pac=new Packet();
                    pac.sourceAddress=ATPGUtil.ipMap.get(nodeName);
                    pac.sourceName=nodeName;
                    pac.fromAddress=ATPGUtil.ipMap.get(nodeName);
                    pac.fromName=nodeName;
                    pac.toAddress=ATPGUtil.ipMap.get(str);
                    pac.toName=str;
                    pac.fromAddress=ATPGUtil.ipMap.get(routerName);
                    pac.fromName=routerName;
                    pac.addLine(routerName);
                    Vector<String> vec1=new Vector<String>();
                    vec1.add(""+packet.fileNumber);
                    vec1.add(""+packet.packetNumber);
                    vec1.add(packet.sourceName);
                    vec1.add(packet.toName);
                    vec1.add(""+packet.toPort);
                    Router.forwardInfoModel.addRow(vec1);
                    sender.sendPacketObject(pac);
                    if(portSelection>=3){
                        portSelection=1;
                    }else{
                        portSelection++;
                    }
                }
            }
        }catch (Exception e) {
            e.printStackTrace();
        }
    }
```
Toolreceiver.java:

```java
public ToolReceiver(int port) {
    receiveError=new ArrayList<Integer>();
    receiveErrorTest=new ArrayList<Integer>();
    resultList=new ArrayList<String>();
    errorMap=new HashMap<String,ArrayList<Packet>>();
    try{
        ss=new ServerSocket(port);
        Thread thread=new Thread(this);
        thread.start();
    }catch (Exception e) {
        e.printStackTrace();
    }

    public void run(){
        //System.out.println("Hello i'm in tool Receiver \n");
        try{
            while(true){
                s=ss.accept();
                ois=new ObjectInputStream(s.getInputStream());
                Object obj=ois.readObject();
                System.out.println("Receive packet");
                if(obj instanceof Packet){
```
```java
} else if(obj instanceof ErrorObject) {
    ErrorObject error = (ErrorObject) obj;
    if(error.packet.type.equalsIgnoreCase("original")) {
        if(!receiveError.contains(error.packet.fileNumber)) {
            receiveError.add(error.packet.fileNumber);
            String nodeName = error.packet.sourceName;
            int portSelection = 1;
            int fileNumber = new Random().nextInt(100);
            for (int i = 0; i < 100; i++) {
                ArrayList<String> neigh = ATPGUtil.neighMap.get(nodeName);
                for (String str : neigh) {
                    Packet pac = new Packet();
                    pac.sourceAddress = ATPGUtil.ipMap.get(nodeName);
                    pac.sourceName = nodeName;
                    pac.fromAddress = ATPGUtil.ipMap.get(nodeName);
                    pac.fromName = nodeName;
                    pac.toAddress = ATPGUtil.ipMap.get(str);
                    pac.toPort = ATPGUtil.portMap.get(str).get(portSelection - 1);
                    pac.toName = str;
                    pac.destinationAddress = error.packet.destinationAddress;
                    pac.destinationName = error.packet.destinationName;
                    pac.fileName = fileNumber;
                    pac.packetNumber = (i + 1);
                    pac.packetData = ATPGUtil.testContent;
                    pac.type = "test";
                    pac.fileSize = ATPGUtil.testContent.length();
                    pac.toolPacketSize = 100;
                    pac.paths = new ArrayList<String>();
                    pac.paths.add(nodeName);
                    sender.sendPacketObject(pac);
                }
            }
        }
    }
}
```

54
} else if (obj instanceof Ticket){
    Ticket ticket=(Ticket)obj;
    Vector<String> vec=new Vector<String>();
    vec.add("-ticket.ticketNumber");
    vec.add(ticket.title);
    vec.add(ticket.sourceName);
    vec.add(ticket.description);
    Tool.ticketListModel.addRow(vec);
} else if (obj instanceof String){
    if (!resultlist.contains(obj)){
        resultlist.add("+obj");
        JOptionPane.showMessageDialog(null,"Test packets sent successfully");
    }
}
} catch (Exception e) {
    e.printStackTrace();
}