International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 07 | July 2020 www.irjet.net p-ISSN: 2395-0072

Traffic Black Hole Detection Framework

B Sai Bhaskar Reddy[1], Saba Farheen N S[2]

[1]Dept. of Electronics and Communication Engineering, R V College, Karnataka, India

[2]Professor, Dept. of Electronics and Communication Engineering, R V College, Karnataka, India

---

Abstract - *Internet backbone and core networks are under continuous shift, striving to keep up with the increasing demand. Capital expenditures of telecom companies on network maintenance hovered at around 15 percent of the total revenue. A common fault in network is packets getting dropped without evident reasons, where pinpointing the cause of failure is a difficult and tedious job. These kinds of packet drop in traffic over internet is called a" Traffic Black Hole" or" Silent Failures". Automating the debugging process of traffic black hole detection is difficult in case of traditional networks, thus requiring manual intervention leading to an increased investment in time and resources. The goal of this project is to automate the debugging process that is generally employed while detecting a black hole. The framework achieved detection of black hole by considering all the required methods that are presently being used and by interpreting the routines as rules for Health Bot. The proposed Traffic black hole detection framework could successfully detect a traffic black hole that is caused by a Ingest connectivity failure. Due to this framework, resources spent on debugging the issue got reduced by 80 percent than the usual.*

Key Words: **Health Bot (HB), Traffic Black Hole, Junos OS, Open Shortest Path First (OSPF).**

1. INTRODUCTION

The term traffic Black hole refers to packet loss in a stream, which can be caused by wide ranges of technical errors. Detecting a traffic black hole is a tedious work which requires a lot of resources. A typical Problem Report (PR) takes almost a week by an engineer to complete it. The Problem detection part takes a major amount of time and resources. More PRs would result in a delayed release of the newer versions of the Junos OS. Automating detection of black holes can save a lot of time and resources, which will improve the efficiency of the workforce and the team to be more agile and responsive. The automated debugging framework developed in this work helps to simplify, quicken the debugging process required to find the traffic blackhole in the network. It employs basic to advanced methods used conventionally by an engineer to find the cause of loss in traffic and reports back. This way it reduces the meticulous and tedious steps.

In [1], author presents an application framework in which declarative specifications of debugging actions are translated into execution monitors which can automatically detect bugs. The approach is non-intrusive with respect to program source code and provides a high level of abstraction for debugging activities. Efficiently monitoring a network requires full observability of each node. The main challenge lies in correlating the monitoring logs coming from all the nodes and reconstructing useful network-wide knowledge such as routing topology or end-to-end performance. Authors in [3] proposes a monitoring framework in which nodes would report events that are significant to the life of the network, following clearly specified semantics. Clear semantics allows maintaining a network model, running on a computer on the side of the network, thus mirroring the state of the real network.[4] suggest an approach to the development of debugging automation tools and soft- ware testing based on precise program behaviour models. A program behaviour model is defined as a collection of events which have two basic binary relations over the events i.e. inclusion and precedence, and this represents the temporal relationship between all the actions. A language for computations over all the event traces is developed which provides a basis for debugging queries, assertion checking, performance measurements and execution profiles. Backbone networks of Internet are under constant load, struggling to keep up with ever increasing demand. The trend of technology change so often that outstrips the deployment of fault monitoring capabilities, that are built into today's IP protocols and routers. Some of the new technologies across network layers, raise the requirement for unanticipated connections and service disruptions that the built-in monitoring systems cannot detect. In such cases, failures in detection may cause data packets to drop silently without triggering any built-in monitoring systems. These kinds of drops are called "silent failures" or "black holes" can cause critical failures in the network. In [10], the authors a present a simple and effective method to detect and diagnose such silent failures. The proposed method uses active measurement between edge routers to raise alarms when-ever end-to-end connectivity is disrupted, regardless of the cause. These alarms feed localisation agents that employ spatial correlation techniques to isolate the root-cause of failure. Using data from two real systems deployed on sections of a tier-I Internet Service Provider network, successfully detects and localises their known black holes.

2. AUTOMATED DEBUGGING FRAMEWORK

Manual Debugging of a problem starts with deeply analysing the given problem, pinpointing the part of the source code leads to the problem. Modifications or new additions needs to be done to fix the problem. Local testing would be done to avoid any compilation errors. The code would be submitted for regression and sanity tests once it gets passed locally. These tests are performed on all available platforms and are run several times. Once the code passes for regression and sanity tests, one needs to commit to merge the changed code. This usually takes a week by an engineer to complete. The Framework aims to automate the debugging process. When submitted a topology, the framework runs all the commands and processes, which are usually run by engineers to figure out the problem and report back to the user on which point of failure leads to the problem.

2.1 Health Bot

Health Bot (HB) is a programmable telemetry-based analytics application. With it, one can diagnose and root cause network issues, detect network anomalies, predict imminent network issues, and create real time remedies for any issues that come up. HB Data Collection Methods are used in order to provide visibility into the state of the network devices, HB first needs to collect their telemetry data and other status information. HB does Data Collection using sensors. HB currently supports the following sensors:

- Native Google Protocol Buffers (GPB)
- OpenConfig
- Syslog
- iAgent (CLI/NETCONF)
- Simple Network Management Protocol (SNMP)

2.1.2 Health Bot-Rules

HB's primarily function is collecting and reacting to telemetry data from network devices. The role of a rule is defining collection of the data, and to react to the data. Defining a rule requires a language that describes several elements, what a rule is and does, where the rule receives data from, a way to filter or manipulate the data, and then a way to react to that data. In software terms, this type of language is called a Domain Specific Language (DSL), i.e., a language that is specific to one domain.

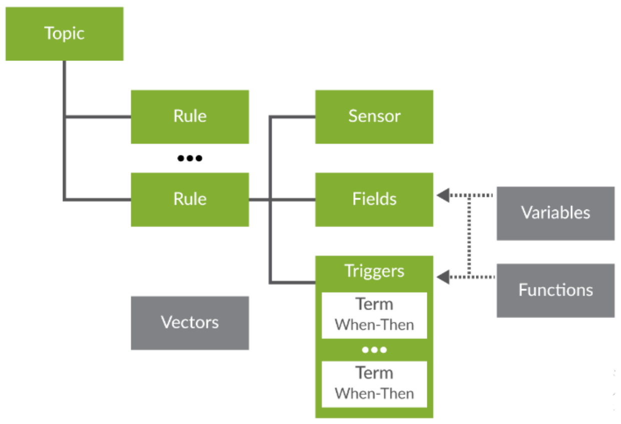


Fig -1: Structure of a Health Bot Rule

HB's DSL includes topics, rules, sensors, fields, triggers, and so on. HB ships with a set of default rules, which can be seen on the Rules page of the HB GUI, as well as in GitHub in the HB-rules repository. One can also create their own rules depending on the requirement.

Sample structure of a HB rule is as shown in Fig. 1. To keep rules organised, HB organises them into topics. Topics can be very general, like system, or they can be more granular, like protocol.bgp. Each topic contains one or more rules. As described above, a rule contains all the details and instructions to define how to collect and handle the data.

2.1.3 Health Bot-Playbooks

In order to fully understand any given problem or situation on a network, it is often necessary to look at a number of different system components, topics, or key performance indicators (KPIs). HB operates on playbooks, which are collections of rules for addressing a specific use case. Playbooks are the HB element that gets applied, or run, on device groups or network groups.

HB comes with a set of pre-defined Playbooks. For example, the system-KPI playbook monitors the health of system parameters such as system-cpu-load-average, storage, system-memory, process-memory, etc. It then notifies the operator or takes corrective action in case any of the KPIs cross pre-set thresholds.

One can create a playbook and include any rules in it. The playbooks need to be applied to the device groups. By default, all rules contained in a Playbook are applied to all of the devices in the device group. There is currently no way to change this behaviour. If playbook definition includes network rules, then the playbook becomes a network playbook and can only be applied to network groups.

2.2 Framework Development

A software framework offers generic software functionality and can be restrictively changed by writing user programs to make it well suited to the required application.

2.2.1 Parsing User's Topology

When a problem is raised by a customer or any other team, the Engineer should again undergo all the steps to reach that problem (here, a traffic black hole). To reproduce the set-up of the customer, the topology information is needed.

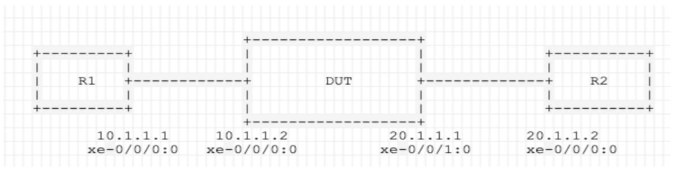


Fig -2: Example network topology

Fig. 2 shows a sample network topology. When a topology is created using pbuilder, a .yaml and a .topo file. The user should input either the topo or yaml file to the framework. The framework should parse the required information as Router's name, Router's ip, Interface info, etc. A python program, topoResolve.py is created to take care of parsing user's topology.

2.2.2 Running required methods

After parsing the required information, all the methods and CLI commands are run on the routers to figure out the traffic black hole. Some of the methods performed are as discussed below.

1. **Device Handle:** A device handle should be acquired on any device to run CLI commands on it. It is done by calling the Device method defined in a standard library.
2. **Interface Statistics:** IF stats give information on packets that have been received and transferred at both physical interface (IFD) and Logical Interface (IFL) stages. This helps to figure out if there has been a drop at any interface, which will help to narrow down on devices.
3. **Stream Statistics and CoS scheduler statistics:** These statistics give information at even deeper levels at the interface. This will help to rule out whether the problem is with the interface hardware or with device control and forwarding plane.
4. **Flexible PIC concentrator (FPC) commands:** All commands required cannot be run on Router Engine (RE), some need to be run on the Forwarding plane. Packet Forwarding Engine (PFE) instances should be known from chassis. A special command needs to be run, for getting information from FPC.
5. **Jnh exceptions:** Jnh exceptions reports on any failure in the FPC. Clearing these will help most of the forwarding plane failures.
6. **Ttrace:** Ttrace gives all the processes a packet underwent. Several steps need to be run on both RE and PFE to get ttrace.

2.2.3 Detection of black hole

Deductions from each method run are different and gives a unique perspective on the problem. All of these can be incorporated into the network. Black holes pertain to those areas of the network where the arriving or departing traffic is silently dropped, without notifying the source that the data is not delivered to the desired destination. The term "Black holes" is given, considering these places are invisible upon inspection of the topology of the network, and can only be detected by examining the lost traffic. All of the above-mentioned methods help to monitor the traffic automatically and make the process of detecting black holes quick and easy.

3. DESIGN OF FRAMEWORK

HB offers several ways to detect and troubleshoot device-level and network- level health problems. The information provided by the subsequent HB GUI pages is used to investigate and discover the root cause of issues detected by HB.

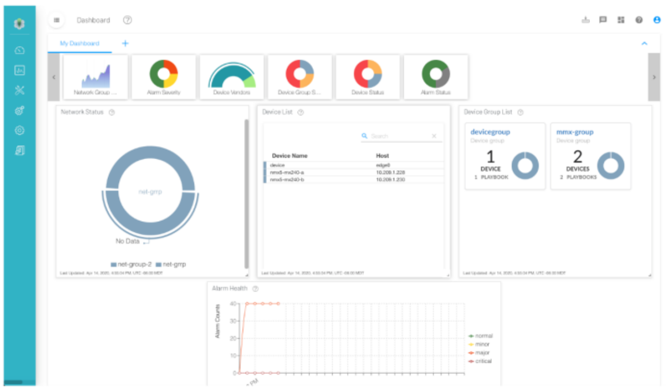
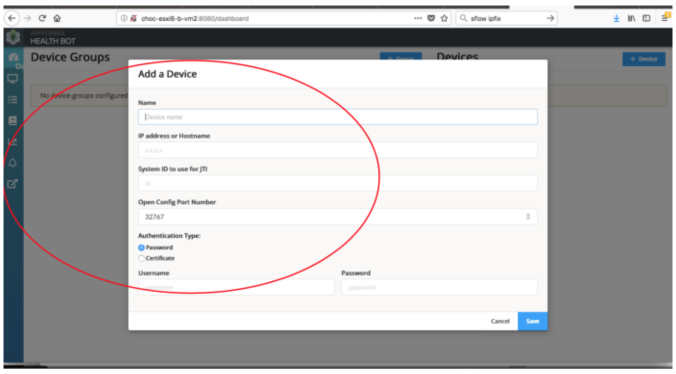


Fig -3 Health Bot Dashboard

The Dashboard also has a graphical list of Pre-defined dash lets across the top that is initially hidden from view. Each dash-let provides graphical information from a specific point of view. Many of the dash lets can be clicked on to drill down deeper into the information presented. The Dashboard is used to create a custom view of one's interests as shown in Fig. 3.

3.1 Adding Devices

The Device information gathered from the user's topology should be added to the HB, that can be done through HB GUI. HB GUI is accessed through port 8080 of the deployed HB.



Add a Device

Name: Device name
IP address or Hostname: x.x.x.x
System ID to use for JTI: id
Open Config Port Number: 32767
Authentication Type: [✓ Password] Certificate
Username: username
Password: password

Fig -4: Adding a Device via HB GUI

Fig. 4 shows the addition of a device using the GUI by choosing the "+Device" tab and filling the information about Device name, IP Address, System ID (needed for native sensors – this document is not covering much on native sensors, but focuses more on Openconfig and iAgent Sensor), Open Config Port Number, Username/ Password. Once the devices are added, they can be grouped logically. This logical grouping is required if the rules/playbooks are to be applied on set of devices.

3.2 Creating Rules

A rule has the details and procedures to collect and handle data. The Sensors defines the attributes for collection of the data. The data collection model is typically used to ingest the data, and to pull or push the data. In a rule, a sensor will be described in the current rule or can be taken from another rule. A yaml file consisting the configurations to the sensor is created. The created yaml file needs to be uploaded in the sensor section. Respective sensor is also needed to be selected.

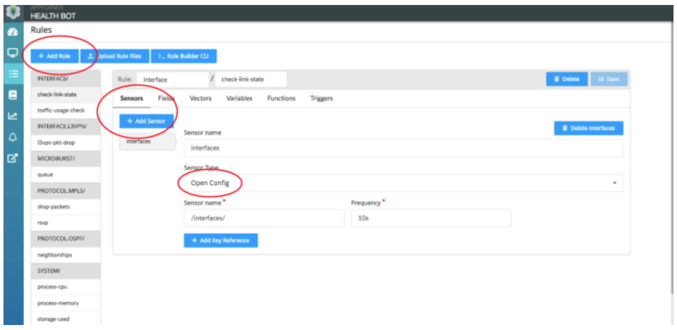
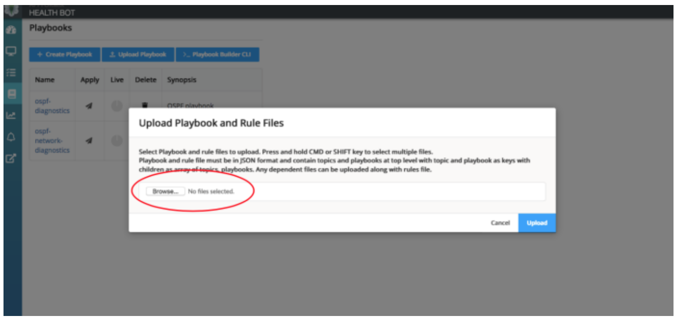


Fig -5: Creating a custom rule.

Fig. 5 shows creation of rule in HB GUI. The sensor usually intakes a huge amount of data, so fields will enable a way to filter or manipulate that data, which allows to identify, modify and facilitate that data. Triggers consistently and periodically brings together the values and fields elements which are used to compare and gather the required status of the system.

3.3 Uploading Playbook

The methods that are created are to be added to HB. Each script is added as a rule and collections of rules become a playbook. The created playbook when run on a given topology will have an ability to detect the black hole.



Upload Playbook and Rule Files

Select Playbook and rule files to upload. Press and hold CMD or SHIFT key to select multiple files.
Playbook and rule file must be in JSON format and contain topics and playbooks at top level with topic and playbook as keys with children as array of topics, playbooks. Any dependent files can be uploaded along with rules file.

No files selected.

Fig -6: Uploading the Playbook

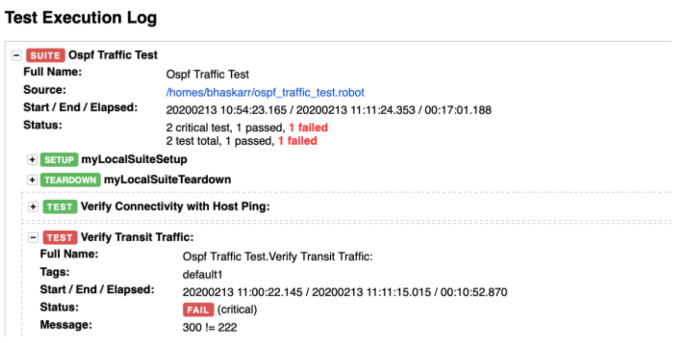
The created system black hole detection playbook should be uploaded to HB. Fig. 6 shows the way to upload a playbook.

4. RESULTS AND DISCUSSIONS

As the need for automation is very much required in the near future and it is populating all over the internet world. This chapter describes the results that are achieved through the system black hole detection HB playbook. A case is considered where a black hole is identified. It details the way HB would manage the traffic black hole detection.

4.1 Traffic Black Hole

A case is considered where packet loss is happening even though every interface is up and forwarding the data streams, a classic Traffic Black hole. The routes are configured by OSPF. Fig. 7 shows that there is a fatal error that is raised due to packet loss.



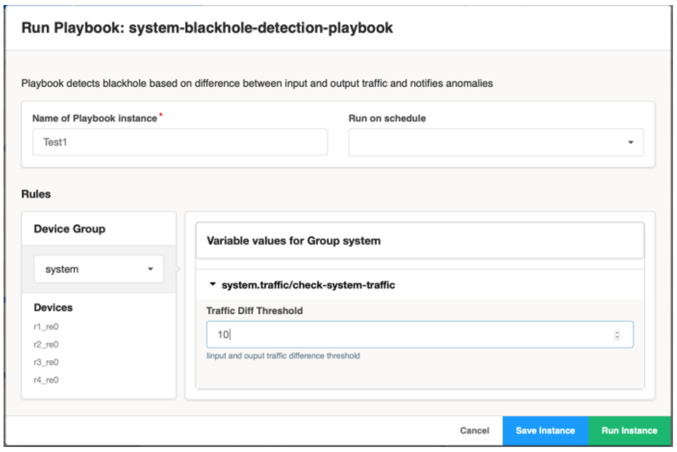
```
Test Execution Log

− SUITE Ospf Traffic Test
  Full Name:            Ospf Traffic Test
  Source:               /homes/bhaskarr/ospf_traffic_test.robot
  Start / End / Elapsed: 20200213 10:54:23.165 / 20200213 11:11:24.353 / 00:17:01.188
  Status:               2 critical test, 1 passed, 1 failed
                        2 test total, 1 passed, 1 failed
  + SETUP myLocalSuiteSetup
  + TEARDOWN myLocalSuiteTeardown
  + TEST Verify Connectivity with Host Ping:
  − TEST Verify Transit Traffic:
    Full Name:            Ospf Traffic Test.Verify Transit Traffic:
    Tags:                 default1
    Start / End / Elapsed: 20200213 11:00:22.145 / 20200213 11:11:15.015 / 00:10:52.870
    Status:               FAIL (critical)
    Message:              300 != 222
```

Fig -7: Toby log, Packet Loss is happening

300 packets were sent from Spirent's transferring port but only 222 were received at the receiving port. Every Interface of the router is up and is not dropping any packets when checked for many iterations. Debugging this manually takes a lot of time and resources which could be used on other important work.

4.2 System Black hole Detection Playbook

A system black hole detection playbook is developed which consists of required routines that are performed by an engineer.



Run Playbook: system-blackhole-detection-playbook

Playbook detects blackhole based on difference between input and output traffic and notifies anomalies

Name of Playbook instance*: Test1

Run on schedule:

Rules

Device Group: system

Devices: r1_re0, r2_re0, r3_re0, r4_re0

Variable values for Group system

system.traffic/check-system-traffic

Traffic Diff Threshold: 10

Iinput and ouput traffic difference threshold

Fig -8: Instantiating Playbook

Fig. 8 shows the playbook being added via HB GUI. The Traffic Difference threshold can be added at the time of initiating the playbook. When saved and deployed the HB starts gathering data from routers via iAgent communication agent and starts running the developed rules.

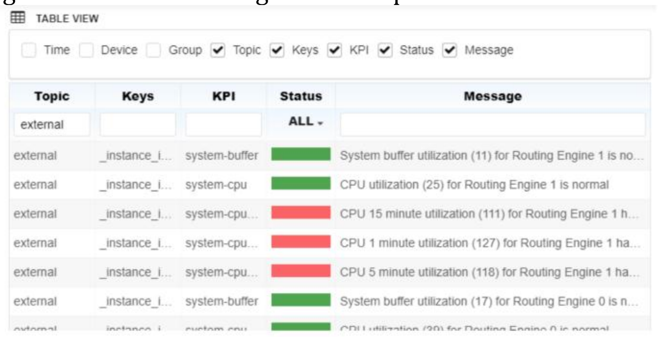


TABLE VIEW

☐ Time ☐ Device ☐ Group ☑ Topic ☑ Keys ☑ KPI ☑ Status ☑ Message

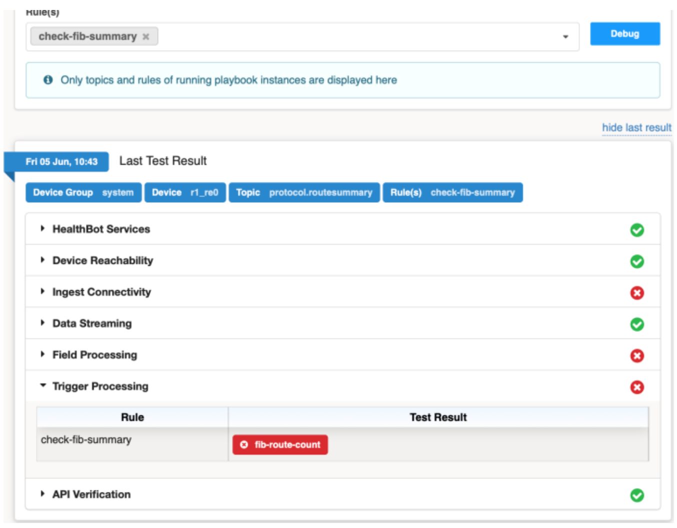
Topic	Keys	KPI	Status	Message
external			ALL	
external	_instance_i...	system-buffer	🟩	System buffer utilization (11) for Routing Engine 1 is no...
external	_instance_i...	system-cpu	🟩	CPU utilization (25) for Routing Engine 1 is normal
external	_instance_i...	system-cpu...	🟥	CPU 15 minute utilization (111) for Routing Engine 1 h...
external	_instance_i...	system-cpu...	🟥	CPU 1 minute utilization (127) for Routing Engine 1 ha...
external	_instance_i...	system-cpu...	🟥	CPU 5 minute utilization (118) for Routing Engine 1 ha...
external	_instance_i...	system-buffer	🟩	System buffer utilization (17) for Routing Engine 0 is n...
external	_instance_i...	system-cpu	🟩	CPU utilization (30) for Routing Engine 0 is normal

Fig -9 Table View

Fig. 9 shows the tabular view of the system resources. The table view gives the progress of the scripts that are running at present, the instances of the resources that are being used while being forwarded, etc.

4.3 Health bot Output

Few seconds after instantiating the playbook, the overview section in HB GUI starts to rise a more specific and relevant error.



Rule(s): check-fib-summary

Only topics and rules of running playbook instances are displayed here

Fri 05 Jun, 10:43 Last Test Result

Device Group system | Device r1_re0 | Topic protocol.routesummary | Rule(s) check-fib-summary

HealthBot Services	✓
Device Reachability	✓
Ingest Connectivity	✗
Data Streaming	✓
Field Processing	✗
Trigger Processing	✗
API Verification	✓

Rule	Test Result
check-fib-summary	✗ fib-route-count

Fig -10: Health bot Result

Fig 10 shows the final test result, a Ingest connectivity issue with check fib rule. Indicating an issue that generally occurs with router tester, Spirent. When dug deeper into the configurations of the spirent and ran multiple tests in the STC, resulted in a thought that there is an issue with the traffic stream. The goal of the original test was to send 300 packets at 10 FPS but the Spirent is configured in a single burst mode which resulted in sending 300 packets at once, i.e., at 300 FPS, which was way more than the original 10 FPS, that resulted in the packets being dropped from RT to DUT.

5. CONCLUSIONS

The aim of this project is to automate tedious debugging process that is usually employed when detecting a black hole. The project has achieved automation of the debugging process by considering all the required routines that are done and by interpreting these routines as rules for HB. Due to this time and resources spent on simulating the user's case, understanding different topologies and running a lot of routines is reduced for the user. Also, it is integrated with the already established HB software which has a great potential in network monitoring and management.

The user's topology information gathering system is developed to simplify understanding and deploying the topology. HB Rules that are required and essential in detecting a traffic Black hole are developed. A system black hole detection HB is created combining the rules. When the playbook is uploaded to HB GUI, it successfully managed in detecting the traffic black hole.

A case is considered with a traffic black hole. The topology information required is acquired through the Topo parser. Devices and Device Group are added to the HB via HB GUI. The system black hole detection framework is uploaded to the HB. The traffic stream got started, after few moments the result section started to show an ingest connectivity issue with the router tester. This playbook helped in understanding, analysing and responding to the issue many times faster than manual debugging process. By using this process 80 percent of the time spent on identifying and debugging the traffic black hole is reduced.

REFERENCES

1. A framework for automatic debugging,Proceedings 17th IEEE International Conference on Automated Software Engineering, Sept. 2002.

2. A Survey Paper on Debugging Tools And Frameworks For Debugging Real Time Industrial Problems And Scenarios,2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN).

3. Sensorlab2: A monitoring framework for IoT networks,2017 International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN).

4. M. Auguston, "Lightweight semantics models for program testing and debugging automation", Proceedings of the 7th Monterey Workshop on "Modeling Software System Structures in a Fast-Moving Scenario", pp. 23-31, June 13–16, 2000.

5. High-level debugging of distributed systems: The behavioral abstraction approach, Peter C.Bates, Jack C.Wileden Volume 3, Issue 4, December 1983.

6. M. Ducasse, "COCA: An automated debugger for C", Proceedings of ICSE 99, pp. 504-513, 1999.

7. M. Matsushita, M. Teraguchi and K. Inoue, "Effective testing and debugging methods and its supporting system with program deltas," Proceedings International Symposium on Principles of Software Evolution, Kanazawa, Japan,2000, pp. 282-289, doi: 10.1109/ISPSE.2000.913249.

8. J. Ji, G. Woo, H. Park and J. Park, "Design and Implementation of Retargetable Software Debugger Based on GDB," 2008 Third International Conference on Convergence and Hybrid Information Technology, Busan, 2008, pp. 737-740, doi: 10.1109/ICCIT.2008.268.

9. D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In Symposium on Operating Systems, Principles, SOSP, pages 131–145, 2001.

10. Luyuan Fang, A. Atlas, F. Chiussi, K. Kompella and G. Swallow, "LDP failuredetection and recovery," in IEEE Communications Magazine, Oct. 2004.