UNIVERSITY OF PIRAEUS

DEPARTMENT OF DIGITAL SYSTEMS
POSTGRADUATE PROGRAMME "DIGITAL COMMUNICATIONS & NETWORKS"

MASTER THESIS

Experimentation platform and algorithm for network and application diagnostics in 5G

Polychronis Pantakis

Supervisor:
Dr. Kostas Tsagkaris, Adjunct Professor

PIRAEUS

JUNE 2021

MASTER THESIS

Experimentation platform and algorithm for network and application diagnostics in 5G

Polychronis Pantakis

MΨE1805

ABSTRACT

5G networks will support demanding services such as enhanced Mobile Broadband, Ultra-Reliable and Low Latency Communications and massive Machine-Type Communications, which will require data rates of tens of Gbps, latencies of few milliseconds and connection densities of millions of devices per square kilometer. In order all these above services to be reliable in 5G networks there is an increase in the interest in software solutions that will help to provide this reliability. Therefore, root cause analysis and performance diagnosis has been gaining popularity in order to find effective methods to provide reliability to the 5G services. These methods consist of two major aspects, prediction and localize faults and service degradations that will help network engineers to make fact-based decisions on how to improve the system or mitigate the possible faults. In this master thesis we implement a performance diagnostics platform which implements an algorithm based on adjacency lists to perform Root Cause Analysis (RCA).

KEYWORDS: Root Cause Analysis, Fault detection, Fault localization, Performance diagnosis, 5G

To my family and

to those who stood by me,

with great appreciation and love.

ACKNOWLEDGMENTS

Upon completion of this master thesis, I would like to thank all who helped to complete this work. First of all, I would like to thank the supervising professor of the University of Piraeus Dr. Kostas Tsagkaris for the help and support he offered to me throughout the implementation period of this thesis.

I would also like to thank Ioannis Chondroulis, Christos Ntogkas and Evangelos Kosmatos for their valuable assistance they offered to me generously during the completion of this master thesis.

Finally, I would like to thank my family and my fiancé for the valuable psychological support they have provided me.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Structure of Master Thesis

We will start by making a reference to the various technologies that will use at the completion of this work so that the reader can understand and be able to get acquainted with them. Then we will describe the RCA algorithm and the topology of the experimental network and finally the analysis, study and presentation of various experimental measurements that will be implemented.

1. Introduction

Computer networks are becoming an essential part of everyday life. Networking systems can be viewed as a key-means for a wide range of services in critical domains including defence, transportation, manufacturing and healthcare.

Meanwhile, impressive changes in the telecommunication industry have emerged thanks to significant evolution of technological science. Wireless and mobile communication technologies have developed massively leading to the ability to connect various wireless technologies, networks, and applications simultaneously. The most recent technology in this field is called 5G. The fifth generation wireless system (or 5G for short) is the newest generation of wireless communication systems. It is the next significant phase of mobile telecommunications standards following the current 4G. 5G moves beyond computer networks for mobile devices alone toward systems that connect various types of devices functioning at significantly higher speeds (Table 1).

Similarly to previous cellular networks, 5G networks consist of cells divided into sectors and send data through radio waves. Each cell is connected to a network backbone through a wired/wireless connection. 5G may transmit data over the unlicensed frequencies that are now used for Wi-Fi. Upon application, it is expected to be an efficient, smart network with unprecedented speed. The target mission of much anticipated 5G is to have by far higher speeds available, at larger capacity per sector, and at lower latency than 4G. In order to optimize network efficiency, the cell is subdivided into micro and pico cells and thus, 5G will be a new mobile revolution as it promises to offer gigabit-per-second data rates whenever and wherever in the entire world [1].

Table 1: Basic comparison among 1G, 2G, 3G, 4G and 5G Technology

Technology	1G	2G	3G	4G	5G
Evolution	1970-1980	1990-2004	2004-2010	2010	2015
Frequency Band	824-894MHz	850-1900MHz	1.8-2.5GHz	2-8GHz	3-300GHz
Speed	2.4Kbps	64Kbps	144kbps- 2Mbps	100Mbps- 1Gbps	Higher than 1Gbps
Signal	Analog	Digital	Digital	Digital	Digital
IEEE Standards	802.11	802.11b	802.11g/a	802.11n	802.11ac
Standards	AMPS,TACS	GSM based, GPRS (50Kbps), EDGE (1Mbps)	UMTS/HSPA	LTE/LTE Advance, WiMax, Wi-Fi	WWWW
Switching	Circuit	Circuit, Packet	Packet except for air interface	All Packet	All Packet
Core Network	PSTN	PSTN	Packet N/W	Internet	Internet
Services	Voice Telephony	Digital voice, SMS, Higher capacity packetized data	Integrated high quality audio, video and data	Dynamic data access, wearable devices	Dynamic information access, wearable devices with AI capabilities.

Additionally to the previously mentioned advantages, 5G is sufficiently capable to support both software and consultancy. It has increased data rate at the edge of the cell and more extensive coverage area. However, the transformation from 4G to 5G is quite challenging due to various steps of transition that have to be fulfilled or tackled to fully realize the 5G vision. Thankfully the technologies that are being developed to enable 5G are expected to face successfully these challenges. However, there are also challenges considering the integration of this new technology to provide services in different application scenarios [2].

A necessary feature of such systems is their sustainability, which can be measured in terms of their ability to tolerate faults and maintain an acceptable performance in the presence of failures.

In order to deal successfully with anomalies that often appear and are unavoidable in communication networks, timely detection, recognition and accurate classification of such errors is pivotal to providing high-level networking services with availability and reliability [3]. The diagnosis of network faults is a quite complex mission. Adequate knowledge of the network architecture and services are much required. This work is usually performed by telecommunication experts who search and analyze network logs to identify issues and determine their origins. Identifying the origin of an anomaly is a key step for efficient troubleshooting. Nevertheless, this process is far from simple. Two identical problems may result in multiple dysfunctions in different points of a network. As a result, experts have to identify these errors and point among them the one creating the issue. With the increased complexity of current networks, this procedure can no longer be executed by human experts, nor can it be deployed by a simple expert system implementing a set of hard coded rules.

The development of an automatic solution for network anomaly detection is difficult due to two major aspects [4]: data handling and integration within the monitoring system. The first aspect is related to data preparation and processing. The metrics of the monitoring system can be either constant, periodic, or chaotic. Since our target here is to address the issue of anomaly detection in periodic data, the solution has to automatically recognize the types of the metrics and process only the periodic ones. Periodicity detection is quite challenging, especially in irregular metrics where the interval between different samples is not constant. This is always the case in real monitoring systems where the data measurements are controlled by queuing systems. Furthermore, the solution has to learn a short history of data as monitoring systems are designed to erase data in short periods. The solution has also to adapt to the natural growth of the traffic. This growth should be integrated into the model of data created by the solution in order to prevent it from being detected as an anomaly. Another challenging fact is the wide range of patterns of anomalies. The solution has to identify new anomaly patterns that were not predefined during the implementation. The second aspect concerns the integration of the solution in the monitoring system. The solution should be transparent and not interfere with any other monitoring process. As a result, it has to have limited computational and memory needs. The solution should not request any post implementation effort from experts. Consequently, all the steps of the process should be fully automatic. Finally, the solution has to provide accurate results which is not easily applicable with limited calculation resources.

As mentioned previously, to achieve efficient diagnosis, it is required to perform a deep analysis of the data logs and determine the root cause of issues. The automation of this process is challenging as both domain knowledge and analysis capabilities from experts are needed. The network architecture is required to diagnose network issues. However, to determine the issue a large effort prior to the installation of the solution is manually required. Moreover, this information has to be changed each time the architecture is modified. To surpass this difficulty, the architecture can be discovered automatically based on the data. However, the inference of such complicated structure may need a lot of calculations. The network architecture is not the only information that is needed. The structure of services provided by Internet Service Providers (ISPs) and web content providers is also required. Additionally, the analysis of communication logs to export information about existing issues is difficult in multiple aspects. The number of communication logs is vast, each having a large number of features. The features may be related. There is no dependency chart defining exactly and reliably the relations between different features. Another challenging point is the large heterogeneity of network issues. Some issues may be interrelated and thus even more difficult to be diagnosed automatically. The network issues do not have the same significance and thus a prioritization task must be integrated within the diagnosis process.
Monitoring functions can be divided into three main categories: performance monitoring, troubleshooting, and planning. There are operators that use numerous different systems, each one implementing a part of the monitoring functions while others use a centralized monitoring system that contains and the three categories. Next, we explain in detail each category of the monitoring functions. Troubleshooting is the most significant function as operators want their services to be continuously available without any disruptions. The troubleshooting process can be divided into four main functions:

- Data collection, issue detection, issue identification, and recovery. Data collection is responsible for collecting data that is relevant to the troubleshooting process by creating and collecting logs, generating reports, mirroring the traffic, recording events, and calculating metrics. Next, issue detection function is responsible for evaluating the several metrics from the previous step and analyzing logs in order to detect any anomalies that may occur. If a metric has an abnormal value or any anomaly is found in the logs, then the operator is notified. The notification the operator gets can have multiple forms such as raising an alarm or creating a ticket. The third function, issue identification consists in a deep analysis of the data to diagnose the network and identify the issue. Finally, the recovery function includes fixing the problem by triggering the adequate compensation and recovery mechanisms. The above four functions of the troubleshooting process can be manual, partially automated, or fully automated.
- Performance monitoring is responsible for measuring Key Performance Indicators (KPIs) to gain insights regarding the network's performance, benchmark network services and elements, and identify the low performing ones. Through performance monitoring, operators may improve the QoS and therefore the QoE.

- Planning and prediction are based on the monitoring of the various resources and processes. By analyzing KPI trends, operators can improve their services in different ways. As an example, they can anticipate events and prevent downtime. Furthermore, they can predict churn and propose more personalized subscriptions. Moreover, they can allocate resources more efficiently.

To ensure network high performance, the monitoring system has to be effective. In order to succeed this effectiveness different requirements must be met. First, the monitoring system should be flexible to supervise multi-vendor and heterogeneous network equipment and infrastructure. Second, the monitoring system should scale to handle the growth of the traffic and the expansion of cellular networks. Third, the monitoring system should be adequate reactive to detect any issues that may occur and trigger mitigation operations in real time to limit downtime. In addition, it should perform an in-depth analysis of the network and go to fine granularity levels to find hidden issues. Fourth, the monitoring system should be autonomous where it is possible and reduce demanded human efforts. Routine tasks should be automated and more advanced tasks can be partially automated by the use of ML. The settings of the monitoring system should be straightforward. Also, it has to be compliant with the several telecommunication standards and market. The monitoring system should respect the standards by insuring the demanded QoS. It has also to respond to the marked needs (e.g. integrating new functions to monitor new services). Furthermore, it should be fault tolerant and easy to troubleshoot. Finally, the monitoring system has to be cost-effective. As the network traffic is huge, the analysis implemented in the monitoring system should be optimized to reduce computational needs. The monitoring system should also store only needed data and for a limited period of time.

While the monitoring systems continuously evolved, they are still not meet the requirements mentioned above in many aspects. First, the monitoring process still relies on the human presence. Many monitoring tasks today are still carried out manually. While monitoring systems generate KPIs and alarms, these latter are analyzed by experts when troubleshooting the network. This has as a result to make the troubleshooting task very costly for the operators. Second, the cellular networks still suffer from low efficiency occasionally (downtime) or continually in some specific cases such as roaming and mobility. The monitoring systems are not sufficiently reactive and efficient to address unavailable services in real time nor to handle roaming and handovers in an optimal way. Last, the 5G standards have set high expectations in terms of QoS. The current monitoring systems are not capable of guaranteeing such performance.

Troubleshooting issues related to network is becoming more and more mission critical every day. In order to successfully identify and solve defects that have an impact on service and lower Mean-Time-To-Resolution (MTTR), ITOps need to monitor a wide range of data and metrics, including data in real time.

Troubleshooting network issues consists of many processes - from the initial understanding that something went wrong to the exact identification of the root of the problem. The better understanding of the correlation between network performance and the problem, the faster the issue can be resolved.

The development of a stable network requires to overcome a variety of challenges such as outages, poor performance, and security issues. However, root cause analysis capabilities may provide an answer to these concerns and offer network monitoring solutions.

Root Cause Analysis (RCA) is undeniably a complex process. But in the past, when networks and IT infrastructure had a basic architecture, the identification of a root cause was not that challenging. It was based either on the LAN or the WAN.

Nowadays, this former simplicity has been replaced with ever greater complexity. Today, LANs are still available, but they often connect one server to many others that also connect resulting to feed wireless access points which are how end users access the network.

Meanwhile, computing is now widely distributed. Applications can be found in-house and in the cloud as well, while some of these may be hybrid so the processing is shared. Additionally the on-premises server is almost always virtualized currently, so this creates many servers out of one and makes it even more difficult to find which VM is causing the issue.

Additionally, today's significant applications serve users in multiple departments and spanning various geographic locations around the globe.

Based on the above, there is no doubt that diagnosis and Root Cause Analysis play a pivotal role in error management as it makes networks able to operate reliably by maintaining performance and availability.

2. Root Cause Analysis

2.1 What is Root Cause Analysis

Root cause analysis (RCA) is a systematic process for recognizing “root causes” of problems or events and an approach to resolve or deal with them.

RCA is based on the fact that an efficient troubleshooting system should focus on identifying the reason and location of the problem rather than “putting out fires all day”. After determining the above, it goes further: RCA deals with the problem so that it never appears again.

At first this method of troubleshooting was implemented in the field of aeronautical engineering, but currently it is applied in every industry, but with mostly and to a great extent in telecommunication networks.

Root cause analysis (RCA) is pivotal to face and restrain any possible failures and errors in networks and systems and to minimize the downtime when it happens. It deals with the identification of the “roots” of errors and has an integral part in network management systems along with monitoring, prediction and reparation.

2.2 Why is Root Cause Analysis necessary?

The current business environment dictates that technology is the lifeblood of most operations. Consequently, it is essential for every business to have a reliable and efficient IT infrastructure. That means that it is much needed to have a well-maintained platform that is capable to act automatically and intelligently and quickly gather the required information to perform further testing and troubleshooting on your IT infrastructure. This is what is known as RCA.

RCA has a wide range of advantages where the more significant are:

- RCA focuses on the reason of the problem and not just the symptoms. It identifies and determines the factors that generated the problem or event.
- The cost spent by catching problems early is significantly decreased by RCA.
- Identifying the problem’s cause as early as possible permits the developers/technicians to maintain an agile environment and drive process improvement.
- RCA shortens time to market: when you find the root of the defect in a timely manner and then you correct it quickly and efficiently, the product is released earlier in the market with less uncaught defects.
- Despite the fact that performing RCA might present as time consuming, the opportunity to eliminate risks and root causes is without any doubt worthwhile.

In conclusion, the advantages of adopting the indicated RCA process to prevent defects are numerous since it significantly reduces development time and cost, increases customer satisfaction, decreased work repetition effort and thus, it decreases cost and improves the quality of the product.

2.3 How to perform Root Cause Analysis

The RCA process is usually divided into five major steps:

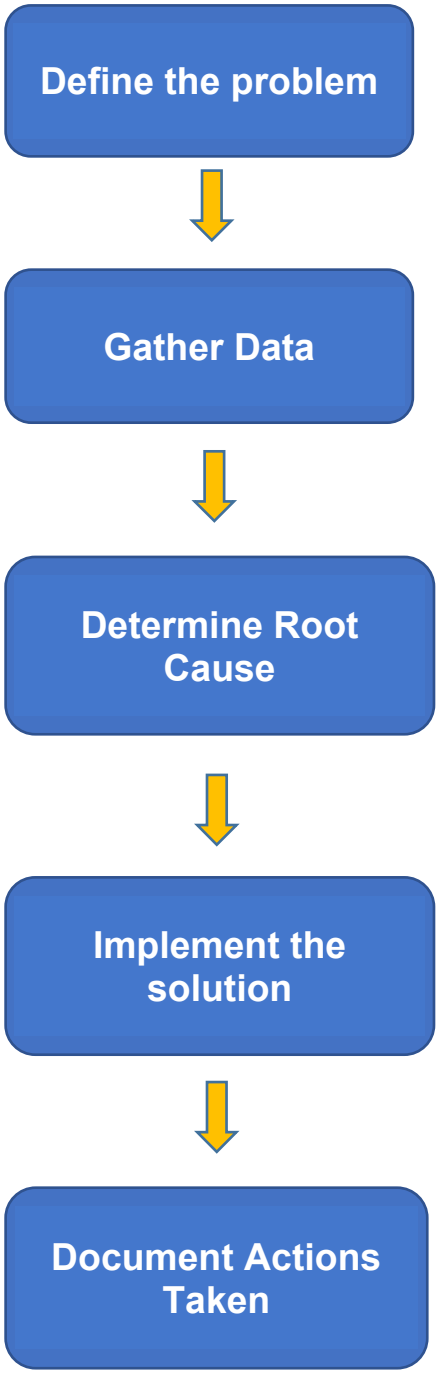

Figure 1: Steps of RCA Process

Let’s analyze the above steps:

- **Define the problem.** When a problem or event appears, the first action you should take is to identify every part of it and try to isolate it. This will help contain the problem.

- **Gather data.** When the problem is defined, gather all data and evidence associated with this specific event in order to understand and identify a possible cause. One significant difficulty in this process of data collecting is to decide the location from which you should start collecting. If the place from which you begin to collect the data is wrong, then you will fail to gather the required data to deal with the problem. On the other hand if you attempt to gather all the available data you may lose again the essential data within the noise that is created. Conclusively a good balance between the two methods is required so that the process runs efficiently.

- **Determine root cause.** This is the true core and main target of root cause analysis. When the sequence of events leading to the appearance of an error have been identified they in turn can be utilized to identify the original root cause of the defect.

- **Implement the solution.** Following the identification of the root cause, one or several solutions will likely be indicated. In some cases you may be able to apply the solution instantly, but in other cases extra work might be needed. Either way, RCA isn’t complete until you’ve applied the indicated solution.

- **Document actions taken.** After you’ve determined and dealt with the root problem, the error must recorded and filed so that engineers can use it as a resource in the future.

3. Platform Architecture

The platform is implemented using ContainerNet [5], the nodes of our virtual network use Ubuntu 18.04 OS and bash scripts are used for the initialization of the platform and for implementing fault injections. The collection of the metrics is performed with ELK stack [6] and a kafka broker [7] is used for getting the metrics from the nodes.

Our platform consists of the below four components

- **Network Configuration component:** This component handles the configuration of the virtual testbed. Initialization and fault injections are performed using bash scripts.

- **Virtual Network component:** This component handles the virtual network over which the performance diagnosis scenarios take place. It is implemented using ContainerNet, a version of Mininet that deploys docker containers instead of virtual hosts.

- **Monitoring component:** This component handles the monitoring and indexing of the metrics. We use the ELK stack for this purpose.

- **Message Broker component:** This component implements a message broker service using Apache Kafka where It collects the metrics.

- **Performance Diagnosis Component**: This component will handle querying the collected metrics/KPIs and the execution of the RCA algorithm

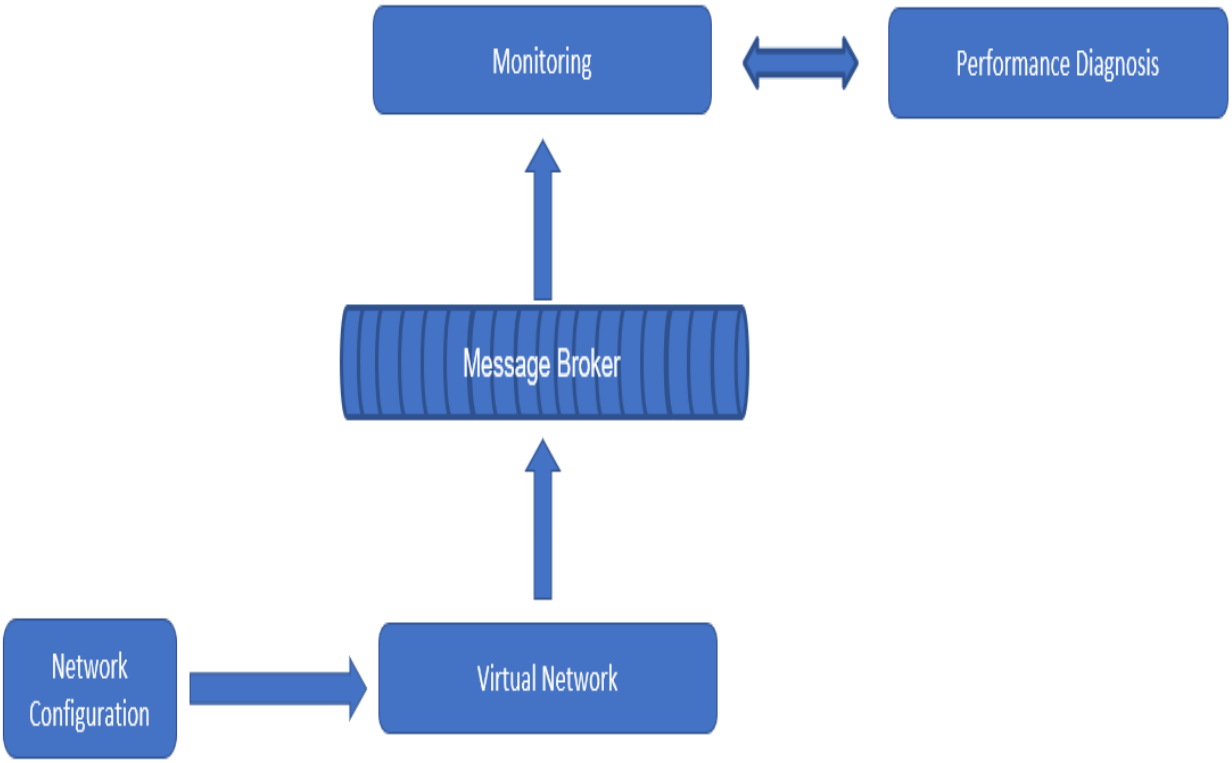

Figure 2: Platform Architecture

3.1 Containernet

Our platform is developed using a tool called Containernet. Containernet extends the Mininet emulation framework and allows us to use standard Docker containers as compute instances within the emulated network. Containernet allows adding and removing containers from the emulated network at runtime, which is not possible in Mininet. This concept allows us to use Containernet like cloud infrastructure in which we can start and stop compute instances (in form of containers) at any point in time. Another feature of Containernet is that it allows to change resource limitations, e.g., CPU time available for a single container, at runtime and not only once when a container is started, like in normal Docker setups.

3.2 Apache Kafka

Apache Kafka [8] is developed by the Apache Software Foundation written in Scala and Java, it is an open source, stream processing platform with aim to provide a unified, high throughput, low-latency platform for handling real-time data feeds.

Kafka is actually a store which gathers data or messages originated by one or many processes that are called producers. The information is then divided in different partitions within various Topic-categories. In each Topic's partition the messages are categorized and gathered together with a timestamp. On the other end, other processes called Consumers can inquire messages from these partitions.

Some popular use cases for Apache Kafka are

- Messaging
- Website Activity Tracking
- Metrics
- Log Aggregation
- Stream Processing
- Event Sourcing
- Commit Log

3.3 The ELK Stack

The ELK platform is a complete log analysis solution, built in combination of three open source tools, Elasticsearch, Logstash and Kibana. ELK uses the open source stack of Elasticsearch for deep search and data analytics. Logstash for centralized logging management and also Kibana for beautiful and powerful data visualizations.

A brief overview of each of these systems follows

Elasticsearch

Elasticsearch is a distributed open source search engine based on Apache Lucence and released under an Apache 2.0 license. It provides reliability, horizontal scalability and multitenant capability for real time search. The searching capabilities are backed by a schema-less Apache Lucene Engine, which allows it to dynamically index data without knowing the structure beforehand. Elasticsearch is able to achieve fast search responses because it uses indexing to search over the texts.

Logstash

Logstash is a data pipeline that enables collecting, defining and analyzing a wide variety of unstructured and structured data and events created across various systems. It provides plugins to connect to various types of input sources and platforms and is helps to successfully handle logs, events and unstructured data sources for distribution into many different outputs with the use of its output plugins or Elasticsearch.

Logstash consists of the following key features

- Centralized data processing
- Support for custom log formats
- Plugin development

Kibana

Kibana is an open source Apache 2.0 licensed data visualization platform that helps in visualizing all types of unstructured or structured data collected in Elasticsearch indexes. Kibana is entirely written in HTML and JavaScript. The highly efficient search and indexing capabilities of Elasticsearch exposed through its RESTful API are used by this platform to display powerful graphics for the end users.

In other words, Kibana makes understanding large volumes of data an easy task. Its simple browser-based interface enables you to quickly create and share dynamic dashboards that display changes to Elasticsearch queries in real time [9].

Figure 3: ELK Stack

4. Our Proposed Algorithm

A typical managed network has a large number of elements and often a large number of fault conditions or abnormalities occur. A single fault in the network may be manifested as multiple alarms in the Network Manage System (NMS). These can result in tens of thousands of alarms getting generated.

Making sense of all the alarms and identifying significant faults can become tedious for the human operator/administrator. It is important to pinpoint the root cause for all the alarms by localizing the fault and generating an alarm only for the faulty elements, as it will reduce the volume of information thrown to the network manager from NMS.

For this reason a low complexity Root Cause Analysis algorithm [10] has been implemented to localize a problematic node that causes performance degradation to other nodes and the deployed service in general.

The basic principle of the algorithm is to check the reachability between the nodes (in our case instances that belong to Virtual Network Functions), using each node's health status that is determined by the network topology and the system's status.

The health status of a node can be dependent on the status of another, non-healthy node in the network path.

The algorithm uses the health status provided, to label the service nodes as "Up" or "Down". An extra label "Unknown" is applied for the "Down" nodes that may be affected by other respective nodes. This is performed using an adjacency list that is obtained from the network topology. Specifically, an n-node undirected graph represented as an adjacency list is created using the virtual links of the topology. The nodes are numbered from 1 to n. The adjacency list is an array (size n) of linked lists where index i of the array contains the linked list of all the nodes directly connected by a network link to node i. Next, each "Unknown" node is examined to identify the Down nodes that may cause the node's non-healthy state. The algorithm's output is one list per "Unknown" node, that contains the "Down" node(s) identified as the root cause for the respective node's performance issues.

Based on the above approach we can split the algorithm in two parts. Part A determines the status of individual elements in the network ("Up", "Down", "Unknown") and Part B creates the Root Cause lists for the "Unknown" nodes.

Specifically,

- **Part A:** Starting with the node connected to the NMS we check it's neighbors (nodes connected by one link). All the neighbors are set as reachable. If a neighbor is Up, it is added to temp buffer. Then we check the neighbors of the rest of the nodes added to the temp buffer. Thus we check every Up node's neighbors. At the end, every node that was reachable and Down is set as unknown.

- **Part B:** We select a target node from the list of unknown nodes (from Part A). We check the status of the neighbors of the target node. If a neighbor is Up it is added to a temp buffer (like in Part A) to be checked next. If it's down, we add it to a set of root causes. After that, we check the neighbors of the first neighbor that was Up, and so forth. In the end we come up with the final set that includes the root cause nodes of the target node.

The flowchart of the algorithm is illustrated in Figures 4 and 5

Figure 4: Part A - Determine Node's Status

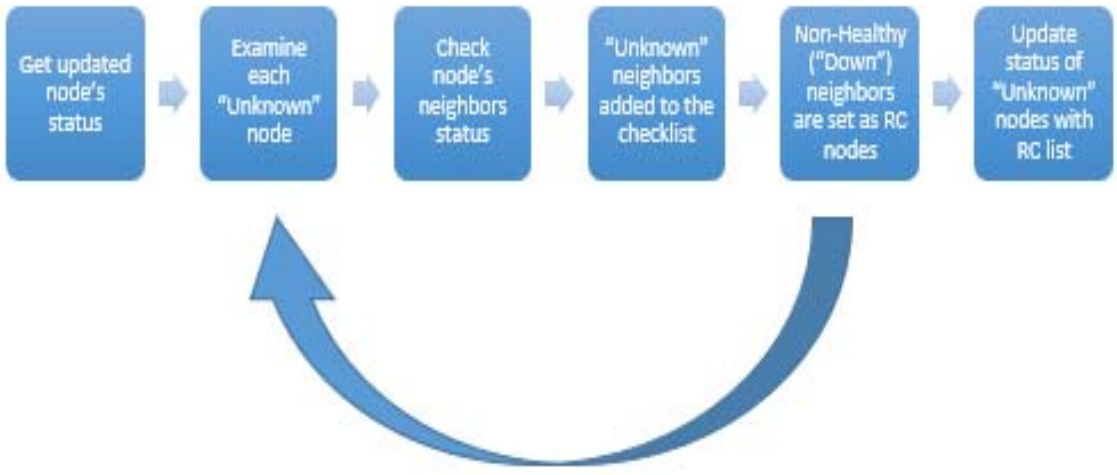

Figure 5: Part B - Find Root Cause Nodes

5. Experimental Scenarios

Below is a description of the experimental procedure as well as the scenarios that were implemented to assess the algorithm's performance.

Purpose of this experimental process is to observe the behavior of the RCA algorithm and evaluate its performance under certain circumstances such as the duration of a fault injection on a node, the time period the impairments algorithm stays in a fault injection command for a node and the algorithm's data collection frequency.

The network topology that was selected to validate the algorithm is a 3-level tree as shown in Figure 6.

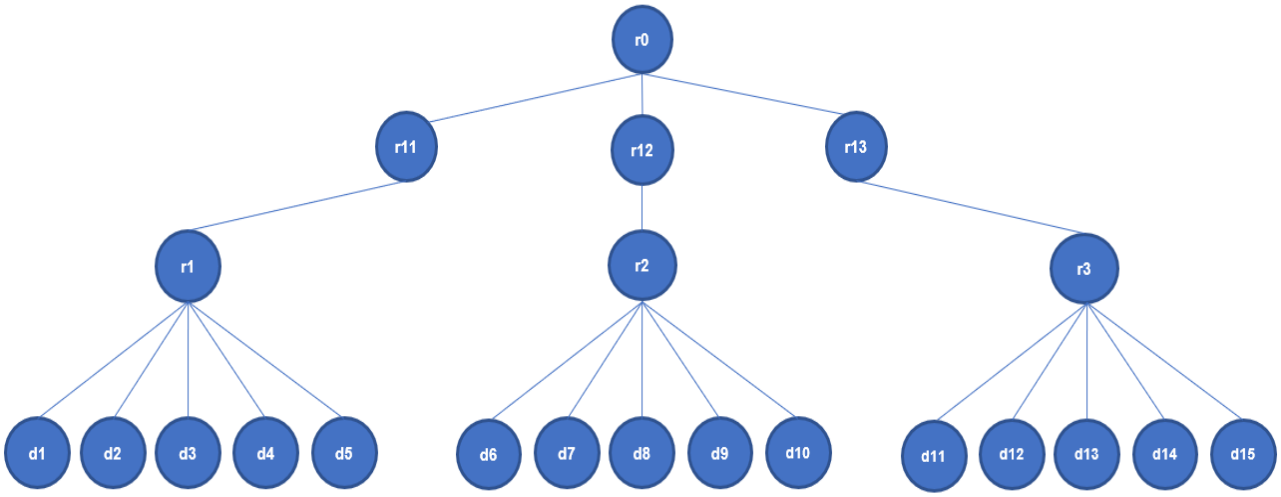

Figure 6: Network Topology

The network topology consists of two types of nodes, the simple nodes (d1 - d15) and the routers (r0,r1,r2,r3,r11,r12,r13).

The fault injection is introduced randomly at the router nodes (marked with red) r1, r2, r3, r11, r12, r13.

The dotted lines show which client is sending data to whom (and waiting for response) over the network producing the "request latency" metric. RAM and CPU overload is injected to the routers and negative impact of that fault injection on the request latency metric is expected

Essentially what we do is to cause impairments to a random router and expect to see increased application latency values at the “clients” sending and receiving requests.

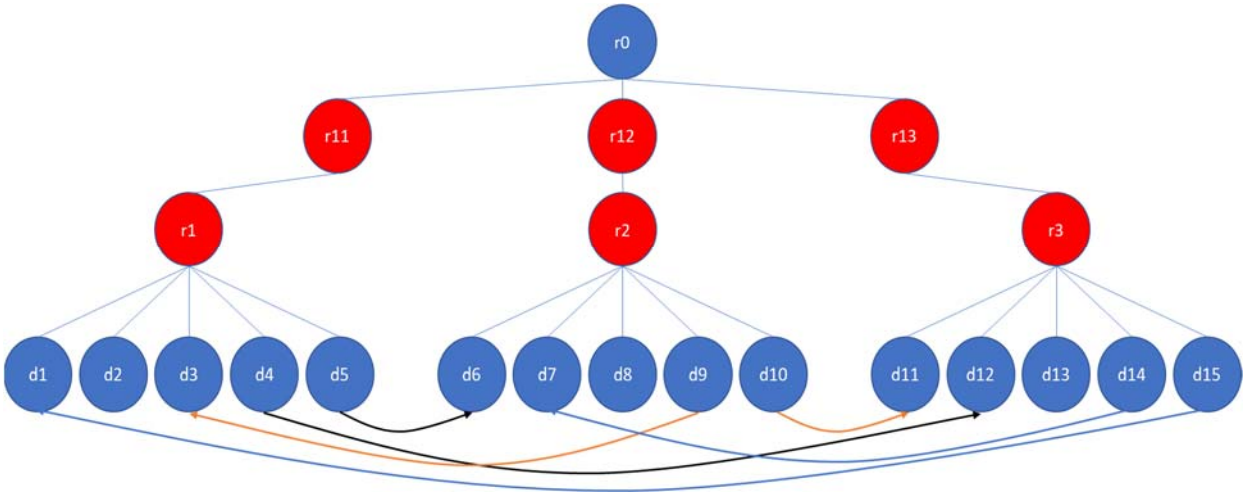

Figure 7: Router nodes with fault injection

In order to test the RCA algorithm’s performance we will use some timing variables which are described below

- Switch Time: The time period the impairments algorithm stays in a fault injection command for a node, until it proceeds to another fault injection command for a different node
- Impairment Time: The time period the fault injection command lasts
- Algorithm data collection frequency: The frequency of time points data is collected by the RCA algorithm

5.1 Experimental Scenarios

- **Scenario 1**: We will evaluate RCA algorithm's performance when the Switch time is greater than the Impairment time
- **Scenario 2:** We will evaluate RCA algorithm's performance when the Impairment time is greater than the Switch time
- **Scenario 3:** We will evaluate RCA algorithm's performance when the Impairment time is greater than 2 times the Switch time
- **Scenario 4:** We will evaluate RCA algorithm's performance when the Impairment time is greater than 3 times the Switch time
- **Scenario 5:** We will evaluate RCA algorithm's performance when the Impairment time is greater than 7 times the Switch time

Our RCA algorithm fetches metrics from the system every 10,15 and 30 seconds for each one of the experimental scenarios described above.

To evaluate the operation of the Root Cause Analysis algorithm, as we have mentioned we will use ContainerNet, a virtual network emulator that utilizes the realistic network emulator of MiniNet (to create virtual controllers, switches and routers) and Docker containers as hosts.

1. As a first step, we built the 3-level topology showed in Figure 6 by running the relevant bash script in the VM where the ContainerNet runs.

```
wings@hwubuntu2:~/chronis$ ./setup_topology_3l_tree.sh
[*] Setting up topology...
```

Figure 8: Bash script for building the experimental topology

Figue 9 below shows that the topology has been created and it is ready to run our experimental scenarios.

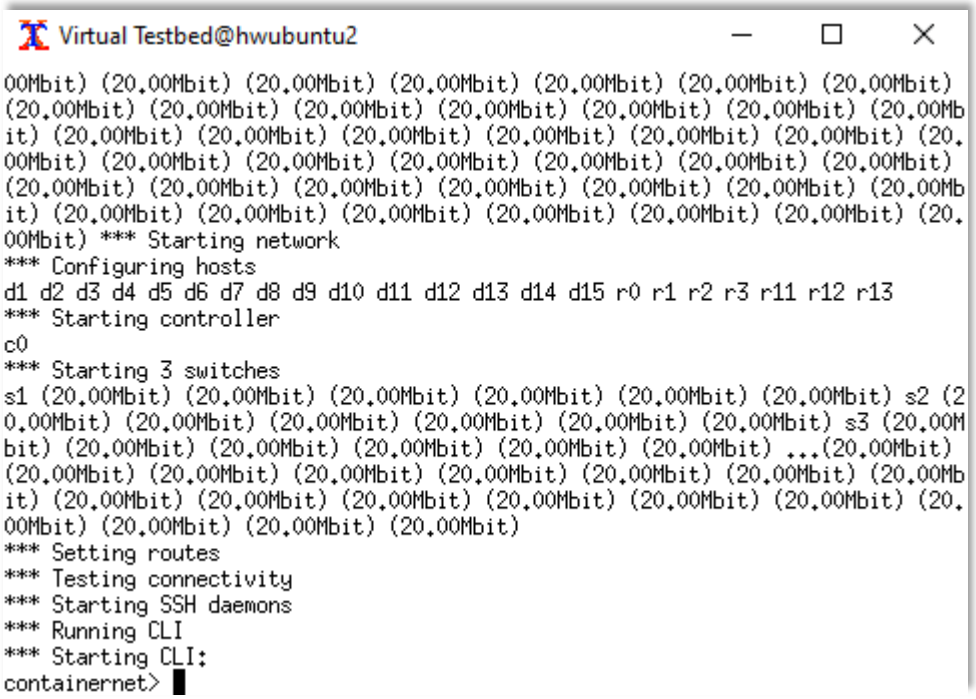

Figure 9: ContainerNet - Topology created

2. On another PC we start running the RCA algorithm. Because we have not add any impairments to the routers the algorithm shows the message “All nodes are up”.

```
C:\Users\Chronis>cd C:\Users\Chronis\Documents\Scenarios\SL20_IM70_R10\3l_tree

C:\Users\Chronis\Documents\Scenarios\SL20_IM70_R10\3l_tree>py main.py
Node r0 | Bandwidth (Tx/Rx): 0.0/0.031 | Latency: 0.02 | App Latency: 0 | CPU: 29.448 | RAM: 24.361 | Disk (R/W): 0.0/4.0
Node d1 | Bandwidth (Tx/Rx): 0.0/0.007 | Latency: 0.03 | App Latency: 0 | CPU: 10.706 | RAM: 22.96 | Disk (R/W): 0.0/4.0
Node d2 | Bandwidth (Tx/Rx): 0.001/0.001 | Latency: 0.04 | App Latency: 0 | CPU: 9.499 | RAM: 23.392 | Disk (R/W): 0.0/4.0
Node d3 | Bandwidth (Tx/Rx): 0.0/0.011 | Latency: 0.03 | App Latency: 0 | CPU: 10.178 | RAM: 23.398 | Disk (R/W): 0.0/4.0
Node d4 | Bandwidth (Tx/Rx): 0.0/0.006 | Latency: 0.02 | App Latency: 363.223 | CPU: 8.612 | RAM: 23.682 | Disk (R/W): 0.0/4.0
Node d5 | Bandwidth (Tx/Rx): 0.0/0.009 | Latency: 0.02 | App Latency: 232.823 | CPU: 8.496 | RAM: 23.048 | Disk (R/W): 0.0/4.0
Node d6 | Bandwidth (Tx/Rx): 0.0/0.011 | Latency: 0.03 | App Latency: 0 | CPU: 10.185 | RAM: 23.399 | Disk (R/W): 0.0/4.0
Node d7 | Bandwidth (Tx/Rx): 0.0/0.008 | Latency: 0.03 | App Latency: 0 | CPU: 10.545 | RAM: 23.57 | Disk (R/W): 0.0/4.0
Node d8 | Bandwidth (Tx/Rx): 0.001/0.001 | Latency: 0.02 | App Latency: 0 | CPU: 11.962 | RAM: 23.579 | Disk (R/W): 0.0/8.0
Node d9 | Bandwidth (Tx/Rx): 0.0/0.008 | Latency: 0.03 | App Latency: 416.096 | CPU: 8.495 | RAM: 23.285 | Disk (R/W): 0.0/4.0
Node d10 | Bandwidth (Tx/Rx): 0.0/0.007 | Latency: 0.03 | App Latency: 289.261 | CPU: 6.693 | RAM: 23.734 | Disk (R/W): 0.0/4.0
Node d11 | Bandwidth (Tx/Rx): 0.0/0.009 | Latency: 0.06 | App Latency: 0 | CPU: 10.028 | RAM: 23.894 | Disk (R/W): 0.0/4.0
Node d12 | Bandwidth (Tx/Rx): 0.0/0.009 | Latency: 0.03 | App Latency: 0 | CPU: 13.306 | RAM: 23.492 | Disk (R/W): 0.0/4.0
Node d13 | Bandwidth (Tx/Rx): 0.001/0.001 | Latency: 0.04 | App Latency: 0 | CPU: 9.393 | RAM: 23.351 | Disk (R/W): 0.0/4.0
Node d14 | Bandwidth (Tx/Rx): 0.0/0.007 | Latency: 0.03 | App Latency: 212.538 | CPU: 8.176 | RAM: 23.497 | Disk (R/W): 0.0/4.0
Node d15 | Bandwidth (Tx/Rx): 0.0/0.009 | Latency: 0.03 | App Latency: 237.865 | CPU: 8.973 | RAM: 23.296 | Disk (R/W): 0.0/4.0
Node r11 | Bandwidth (Tx/Rx): 0.0/0.02 | Latency: 0.02 | App Latency: 0 | CPU: 18.843 | RAM: 24.004 | Disk (R/W): 0.0/4.0
Node r12 | Bandwidth (Tx/Rx): 0.0/0.033 | Latency: 0.02 | App Latency: 0 | CPU: 18.792 | RAM: 24.002 | Disk (R/W): 0.0/4.0
Node r13 | Bandwidth (Tx/Rx): 0.0/0.033 | Latency: 0.02 | App Latency: 0 | CPU: 17.201 | RAM: 23.799 | Disk (R/W): 0.0/4.0
Node r1 | Bandwidth (Tx/Rx): 0.0/0.032 | Latency: 0.02 | App Latency: 0 | CPU: 21.81 | RAM: 24.425 | Disk (R/W): 0.0/4.0
Node r2 | Bandwidth (Tx/Rx): 0.0/0.032 | Latency: 0.02 | App Latency: 0 | CPU: 18.214 | RAM: 24.95 | Disk (R/W): 0.0/4.0
Node r3 | Bandwidth (Tx/Rx): 0.0/0.022 | Latency: 0.02 | App Latency: 0 | CPU: 21.055 | RAM: 24.005 | Disk (R/W): 0.0/4.0
All nodes are up

Indicative timestamp: 2021-06-28 06:24:33



Total runs: 1 | All healthy: 1, No unknowns: 0 No route cause nodes: 0, RC found for: 0
```

Figure 10: Run RCA algorithm with no impairments

3. Next we run the relative bash script which will start the fault injections on the routers of our 3-level tree topology randomly.

```
wings@hwubuntu2:~/chronis$ ./random_impairments_3l_tree.sh
Chosen node: r2
Chosen node: r12
Chosen node: r11
```

Figure 11: Run impairments bash script

4. Returning back to the RCA algorithm, now we observe that there are some SLA violations and the algorithm has detected two root causes.

```
C:\Users\Chronis>cd C:\Users\Chronis\Documents\Scenarios\SL20_IM70_R10\3l_tree

C:\Users\Chronis\Documents\Scenarios\SL20_IM70_R10\3l_tree>py main.py
Node r0 | Bandwidth (Tx/Rx): 0.0/0.038 | Latency: 0.02 | App Latency: 0 | CPU: 14.499 | RAM: 26.437 | Disk (R/W): 0.0/4.0
Node d1 | Bandwidth (Tx/Rx): 0.0/0.003 | Latency: 0.03 | App Latency: 0 | CPU: 8.336 | RAM: 24.648 | Disk (R/W): 0.0/4.0
Node d2 | Bandwidth (Tx/Rx): 0.001/0.001 | Latency: 0.04 | App Latency: 0 | CPU: 7.059 | RAM: 24.423 | Disk (R/W): 0.0/4.0
Node d3 | Bandwidth (Tx/Rx): 0.0/0.012 | Latency: 0.03 | App Latency: 0 | CPU: 7.692 | RAM: 24.835 | Disk (R/W): 0.0/4.0
Node d4 | Bandwidth (Tx/Rx): 0.0/0.004 | Latency: 0.03 | App Latency: 10808.34 | CPU: 5.575 | RAM: 24.237 | Disk (R/W): 0.0/4.0
Node d5 | Bandwidth (Tx/Rx): 0.0/0.01 | Latency: 0.03 | App Latency: 276.636 | CPU: 7.381 | RAM: 24.49 | Disk (R/W): 0.0/4.0
Node d6 | Bandwidth (Tx/Rx): 0.0/0.012 | Latency: 0.03 | App Latency: 0 | CPU: 8.077 | RAM: 24.492 | Disk (R/W): 0.0/4.0
Node d7 | Bandwidth (Tx/Rx): 0.0/0.004 | Latency: 0.04 | App Latency: 0 | CPU: 10.426 | RAM: 24.678 | Disk (R/W): 0.0/4.0
Node d8 | Bandwidth (Tx/Rx): 0.001/0.001 | Latency: 0.03 | App Latency: 0 | CPU: 7.699 | RAM: 24.478 | Disk (R/W): 0.0/4.0
Node d9 | Bandwidth (Tx/Rx): 0.0/0.01 | Latency: 0.03 | App Latency: 205.973 | CPU: 6.425 | RAM: 24.574 | Disk (R/W): 0.0/4.0
Node d10 | Bandwidth (Tx/Rx): 0.0/0.01 | Latency: 0.03 | App Latency: 10813.331 | CPU: 7.105 | RAM: 24.545 | Disk (R/W): 0.0/4.0
Node d11 | Bandwidth (Tx/Rx): 0.0/0.012 | Latency: 0.03 | App Latency: 0 | CPU: 6.382 | RAM: 25.014 | Disk (R/W): 0.0/4.0
Node d12 | Bandwidth (Tx/Rx): 0.0/0.012 | Latency: 0.03 | App Latency: 0 | CPU: 8.97 | RAM: 24.927 | Disk (R/W): 0.0/4.0
Node d13 | Bandwidth (Tx/Rx): 0.001/0.001 | Latency: 0.04 | App Latency: 0 | CPU: 8.234 | RAM: 24.562 | Disk (R/W): 0.0/4.0
Node d14 | Bandwidth (Tx/Rx): 0.0/0.007 | Latency: 0.03 | App Latency: 10208.487 | CPU: 9.515 | RAM: 24.777 | Disk (R/W): 0.0/4.0
Node d15 | Bandwidth (Tx/Rx): 0.0/0.01 | Latency: 0.03 | App Latency: 9643.719 | CPU: 7.282 | RAM: 24.455 | Disk (R/W): 0.0/4.0
Node r11 | Bandwidth (Tx/Rx): 0.0/0.02 | Latency: 0.02 | App Latency: 0 | CPU: 16.084 | RAM: 25.558 | Disk (R/W): 0.0/8.0
Node r12 | Bandwidth (Tx/Rx): 0.0/0.036 | Latency: 0.02 | App Latency: 0 | CPU: 16.657 | RAM: 25.061 | Disk (R/W): 0.0/8.0
Node r13 | Bandwidth (Tx/Rx): 0.0/0.036 | Latency: 0.02 | App Latency: 0 | CPU: 100.0 | RAM: 98.329 | Disk (R/W): 0.0/4.0
Node r1 | Bandwidth (Tx/Rx): 0.0/0.023 | Latency: 0.02 | App Latency: 0 | CPU: 15.777 | RAM: 25.096 | Disk (R/W): 0.0/4.0
Node r2 | Bandwidth (Tx/Rx): 0.0/0.021 | Latency: 0.02 | App Latency: 0 | CPU: 17.84 | RAM: 25.233 | Disk (R/W): 0.0/4.0
Node r3 | Bandwidth (Tx/Rx): 0.0/0.03 | Latency: 0.02 | App Latency: 0 | CPU: 18.6 | RAM: 25.485 | Disk (R/W): 0.0/4.0
Down nodes:

Node: d4 - SLA violation

Node: d10 - SLA violation

Node: d14 - SLA violation

Node: d15 - SLA violation

Node: r13 - non-healthy


NMS node r0

Indicative timestamp: 2021-06-28 05:44:55

The unknown nodes are:

Node d14

The root cause nodes for the target node d14 are:

Node r13
```

```
(non-healthy)

Node d15

(SLA violation)

Node d15

The root cause nodes for the target node d15 are:

Node r13

(non-healthy)

Node d14

(SLA violation)

Total runs: 1 | All healthy: 0, No unknowns: 0 No route cause nodes: 0, RC found for: 2

Rerunning in 10s...
```

Figure 12: RCA algorithm - Root Cause found

6. Presentation of Experimental Measurements

In this chapter we will present and analyze the measurements of experimental process, which will be done through graphs as well as we will comment on the results of these experimental measurements.

6.1 Scenario 1

In this first scenario the algorithm starts collecting metrics sent to the Kafka broker at an indefinite time point, and keeps a certain collecting frequency after that, which is independent from the fault injection timing.

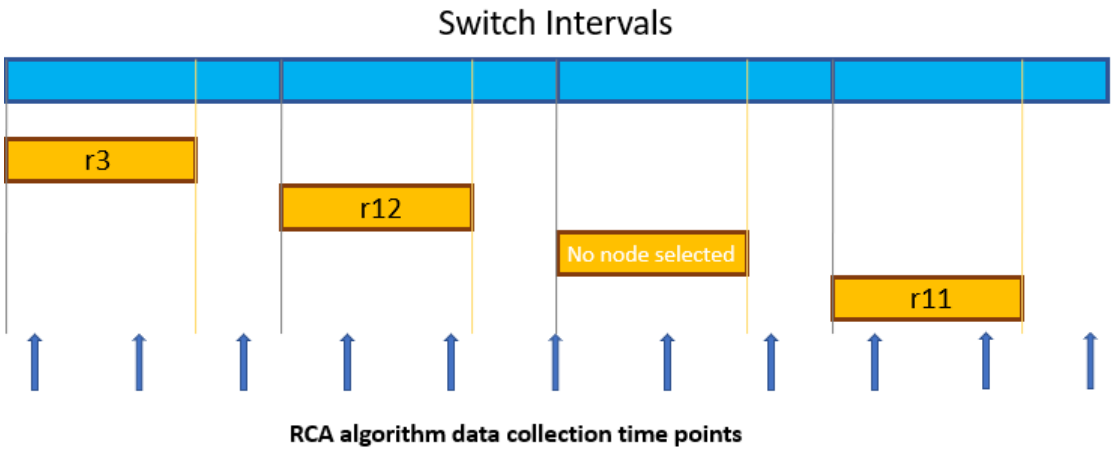

Figure 13: Scenario 1 - Time Evolution

For the experimental measurement of the first scenario we have set the time variables as Table 2 indicates and the results are showed in Figures 14, 15, 16 and 17.

Table 2: Scenario 1 Time Variables

Switch Time	Impairment Time	Algorithm data collection frequency
35 Sec.	25 Sec.	10, 15, 30 Sec.

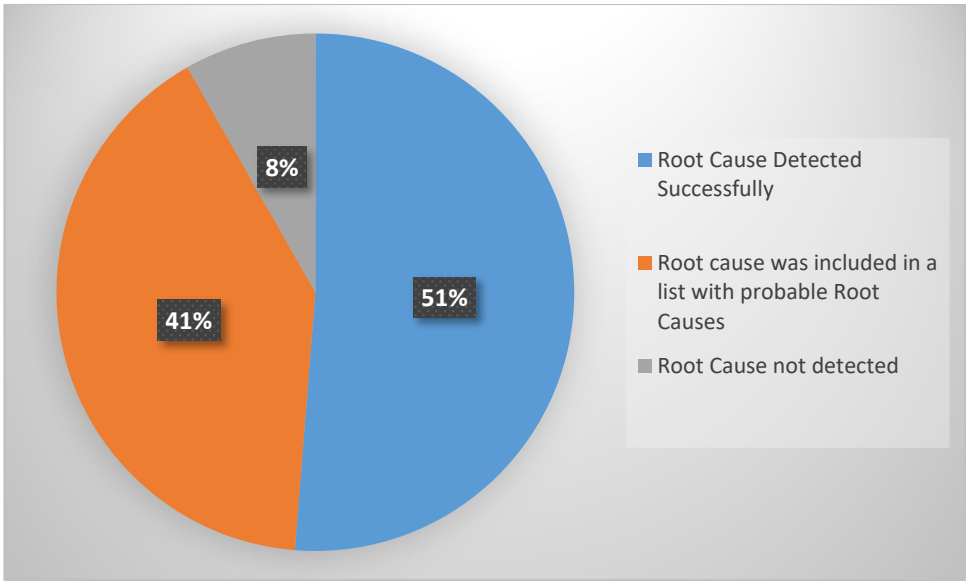

Figure 14: Scenario 1 - Data Collection Frequency 10 Sec.

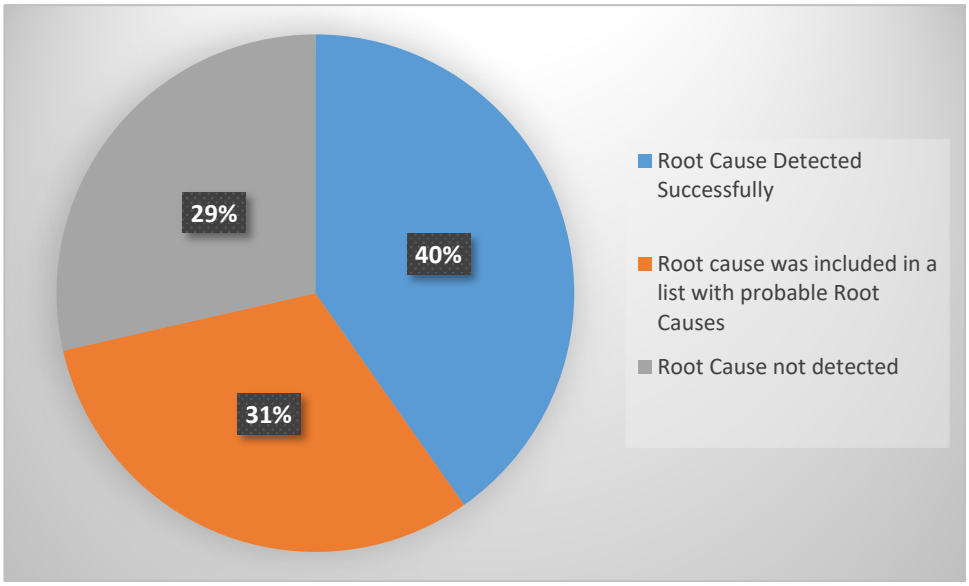

Figure 15: Scenario 1 - Data Collection Frequency 15 Sec.

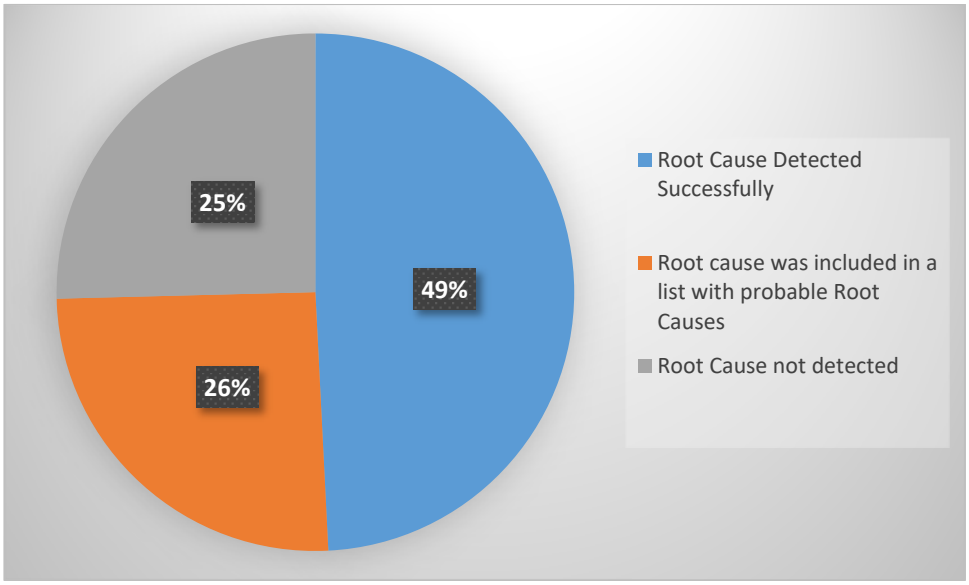

Figure 16: Scenario 1 - Data Collection Frequency 30 Sec.

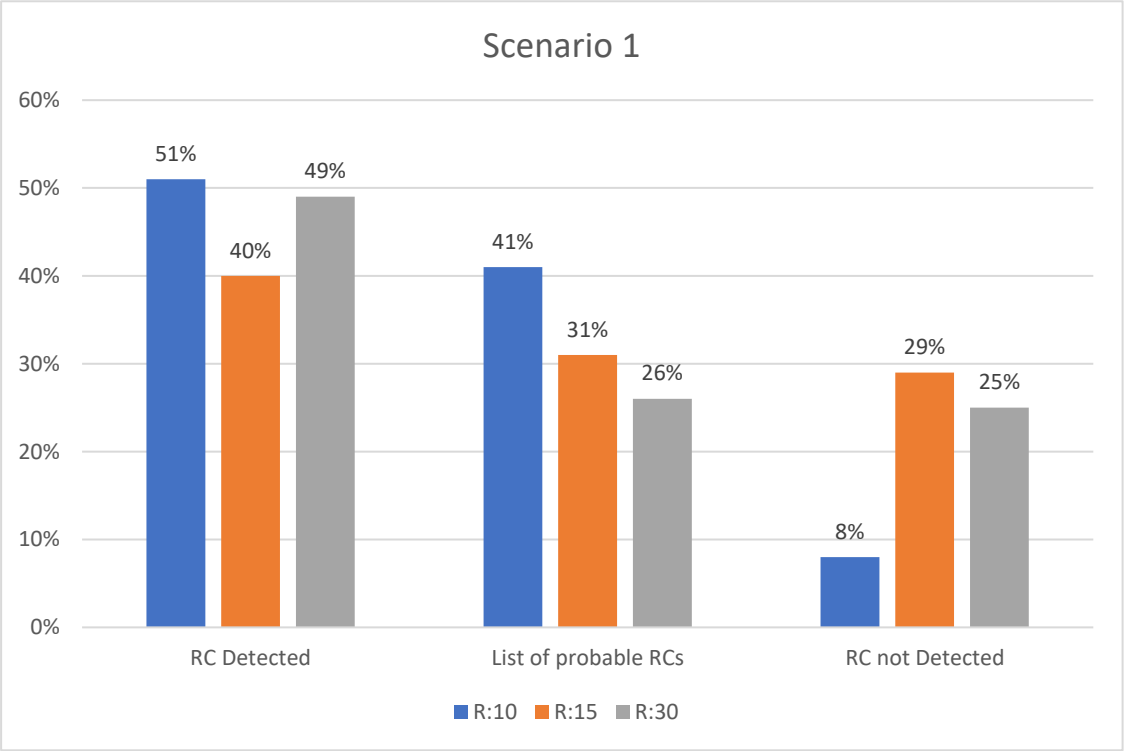

Figure 17: Scenario 1 overall results

From the above results it was found that when the algorithm's data collection frequency is set to 10 seconds the overall accuracy reached a percentage of 92%, meaning that the node responsible for the high response time value was either identified, or included in a list of possible root cause nodes.

When we increased the algorithm's data collection frequency to 15 seconds, the overall accuracy dropped to 71% while for 30 seconds data collection frequency the overall accuracy retained almost the same to 75%.

6.2 Scenario 2

At the second scenario the time variables have been set with the Impairment time to be greater than the Switch time as Table 3 indicates.

Table 3: Scenario 2 Time Variables

Switch Time	Impairment Time	Algorithm data collection frequency
20 Sec.	35 Sec.	10, 15, 30 Sec.

Because of the timing configuration Fault Injection overlaps are produced. Impairments are introduced at 2 nodes simultaneously for specific time points as we can observe in Figure 18.

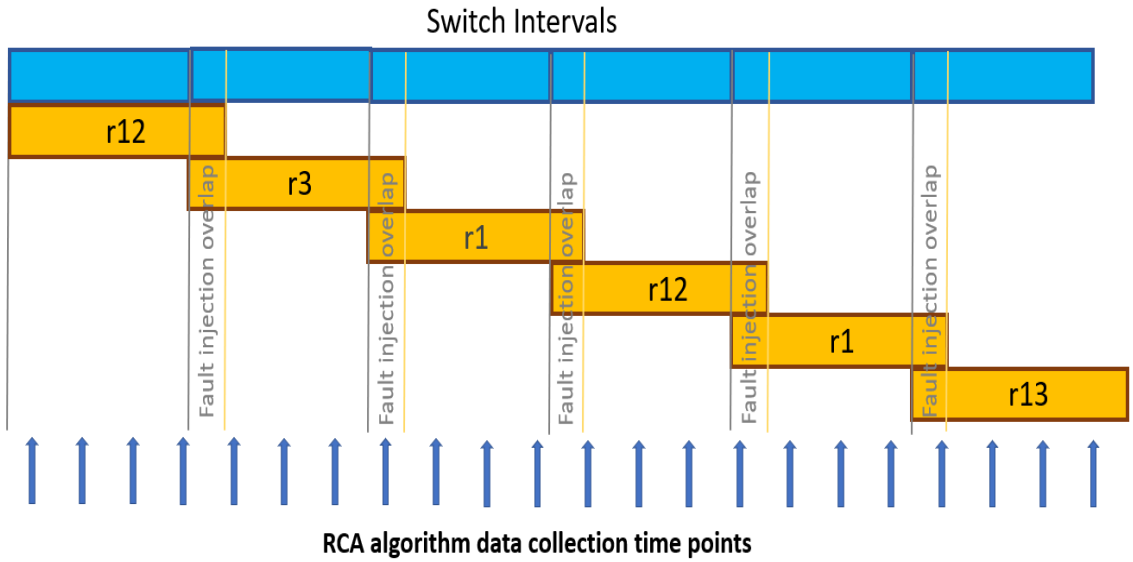

Figure 18: Scenario 2 - Time Evolution

The results of the experimental measurement of the second scenario are showed below.

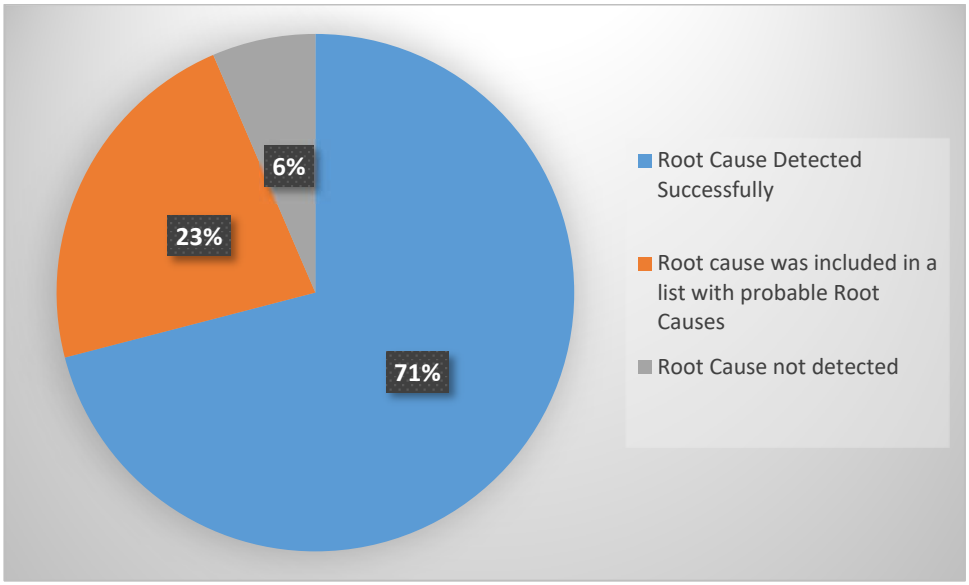

Figure 19: Scenario 2 - Data Collection Frequency 10 Sec.

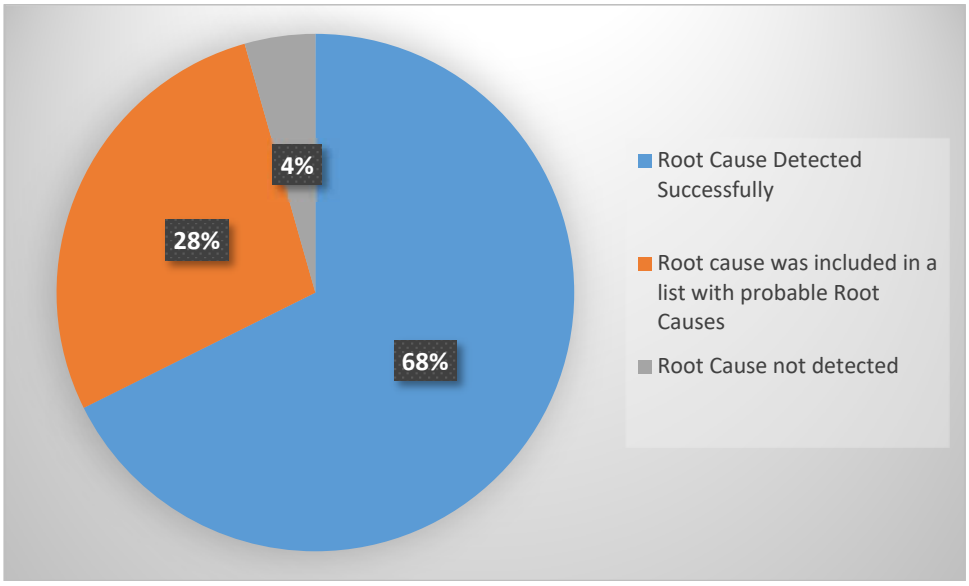

Figure 20: Scenario 2 - Data Collection Frequency 15 Sec.

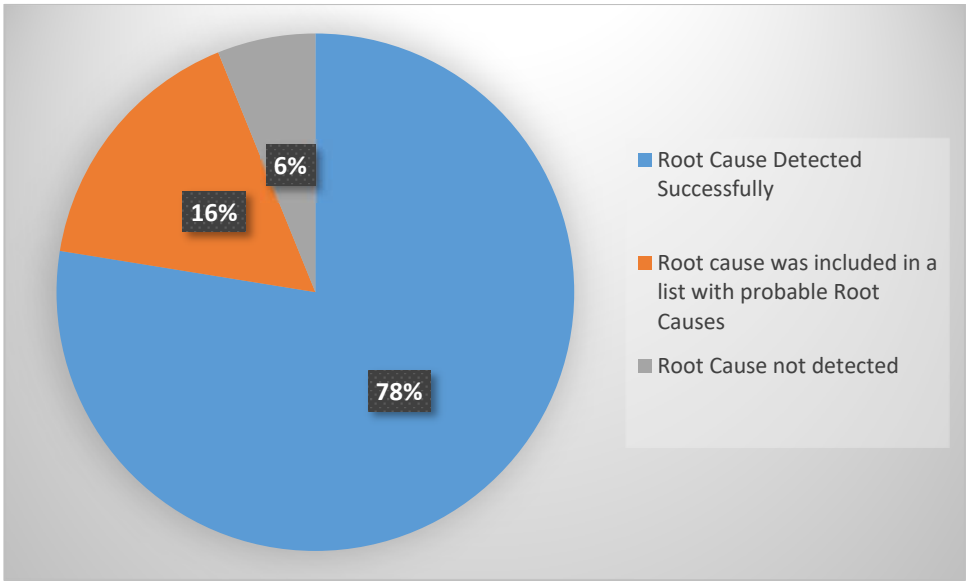

Figure 21: Scenario 2 - Data Collection Frequency 30 Sec.

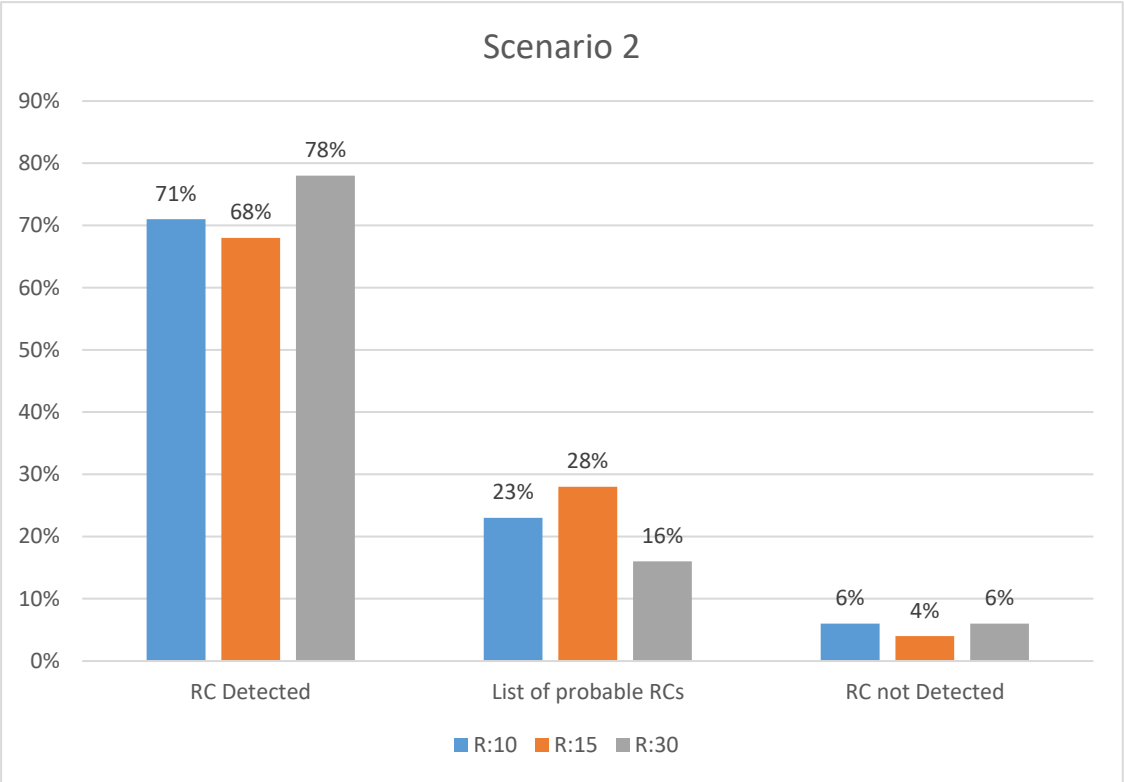

Figure 22: Scenario 2 overall results

Based on the above results we observe much more better performance of the algorithm in comparison with the previous scenario. In this scenario the overall accuracy reached 94% for data collection frequency of 10 seconds, 96% for data collection frequency of 15 seconds and 94% for data collection frequency of 30 seconds.

6.3 Scenario 3

At the third scenario the time variables have been set with the Impairment time to be greater than two times the Switch time as Table 4 shows below.

Table 4: Scenario 3 Time Variables

Switch Time	Impairment Time	Algorithm data collection frequency
20 Sec.	55 Sec.	10, 15, 30 Sec.

Because of the timing configuration multiple overlaps are produced. Impairments are introduced at 2 or 3 nodes simultaneously for specific time points.

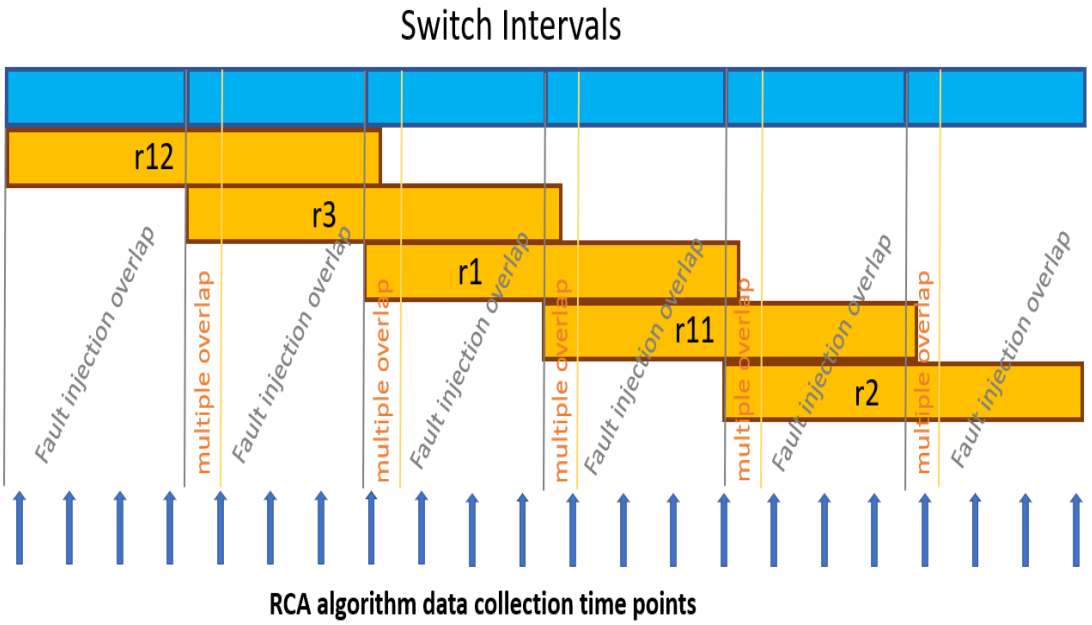

Figure 23: Scenario 3 - Time Evolution

The results of the experimental measurement of the second scenario are showed in the below Figures.

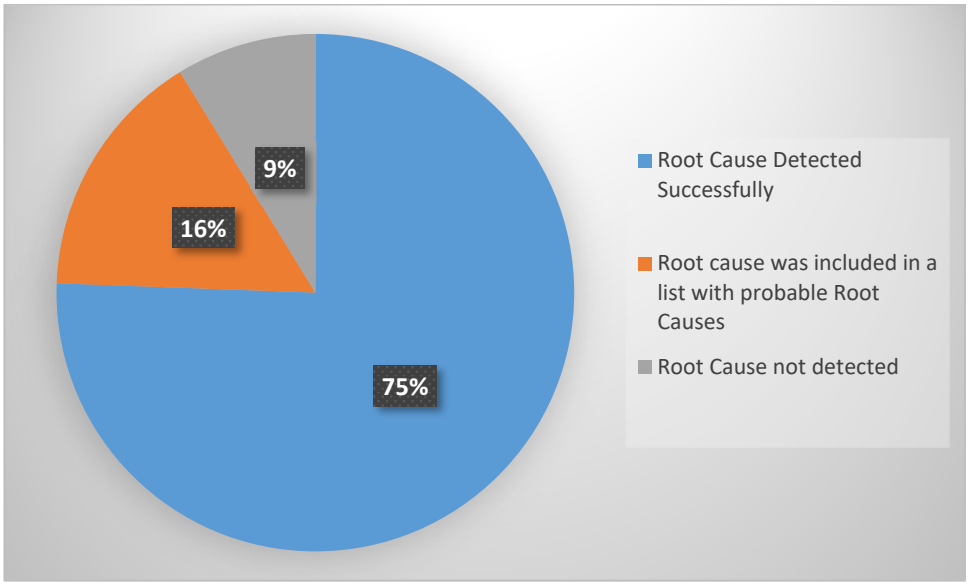

Figure 24: Scenario 3 - Data Collection Frequency 10 Sec.

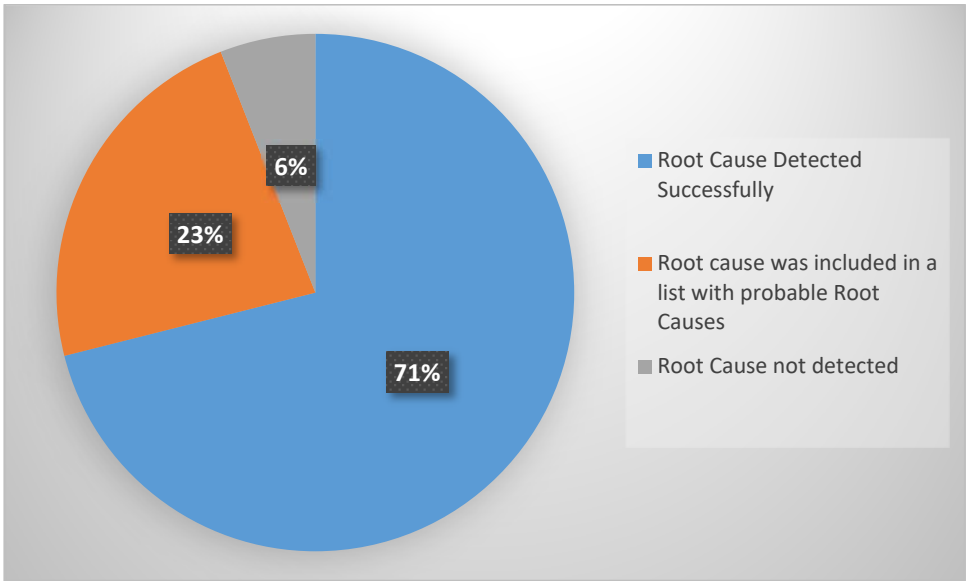

Figure 25: Scenario 3 - Data Collection Frequency 15 Sec.

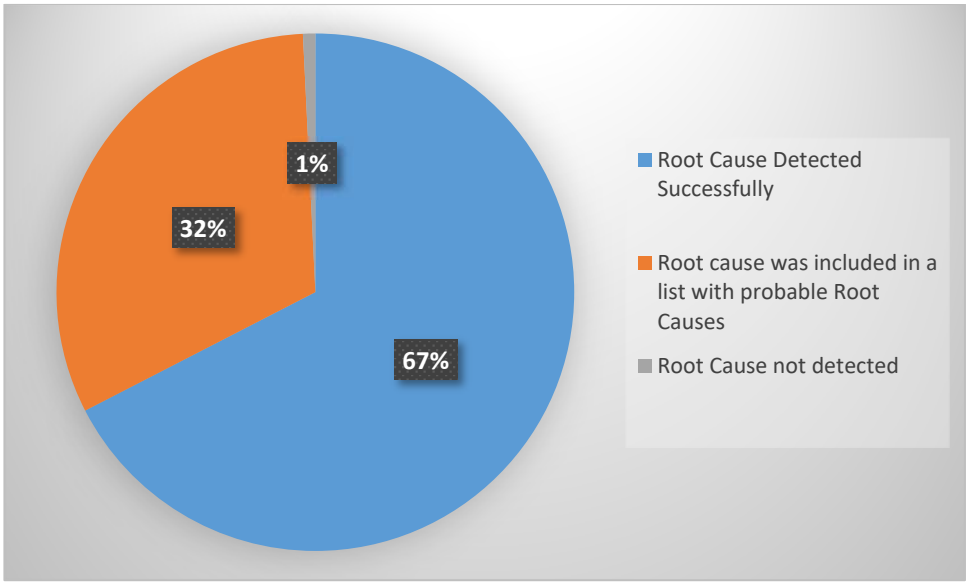

Figure 26: Scenario 3 - Data Collection Frequency 30 Sec.

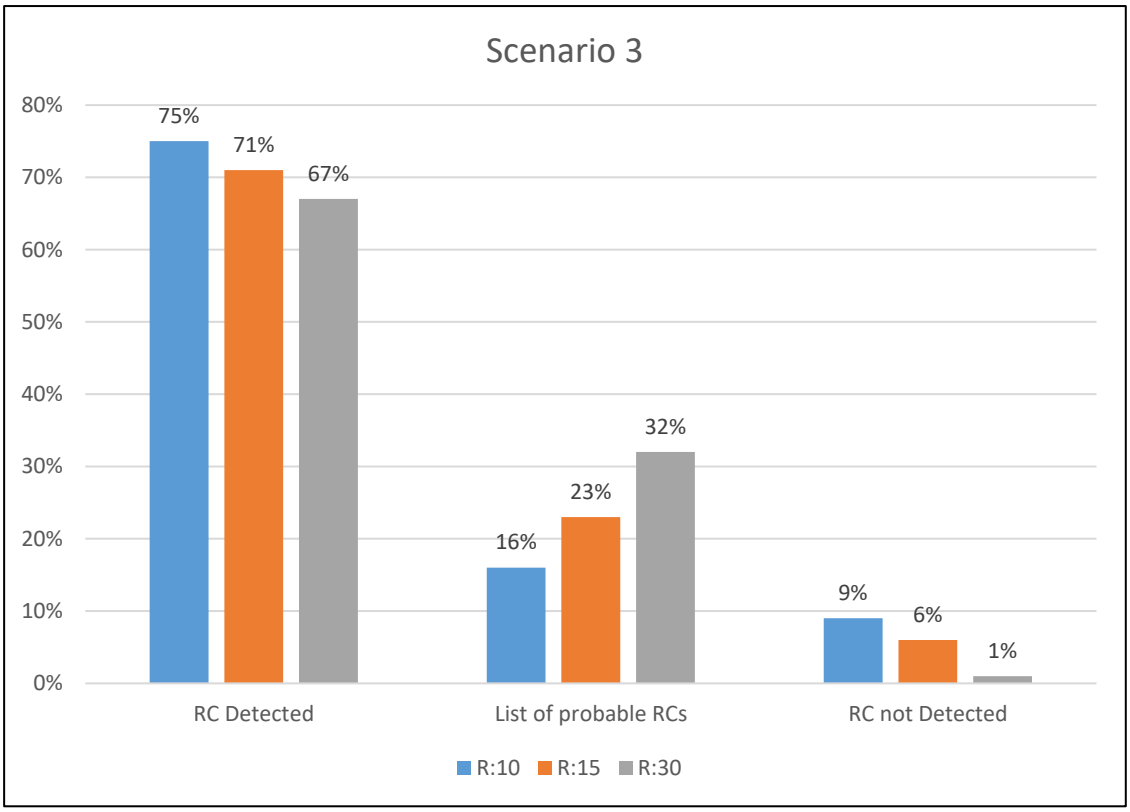

Figure 27: Scenario 3 overall results

Examining the results of the experimental scenario 3 we observe the overall accuracy reached 91% when the algorithm is fetching metrics form the system every 10 seconds, 94% when the data collection frequency is 15 seconds and 99% for data collection frequency of 30 seconds.

Compared with scenario 2, despite the fact that we have multiple fault injection overlaps on the routers the algorithm's performance remains almost the same.

6.4 Scenario 4

At the fourth scenario we continue increase the Impairment time. The time variables have been set for this experimental scenario as Table 5 shows below.

Table 5: Scenario 4 Time Variables

Switch Time	Impairment Time	Algorithm data collection frequency
20 Sec.	70 Sec.	10, 15, 30 Sec.

Because of the timing configuration multiple overlaps are produced same as the Scenario 3. Impairments are introduced at 3 or 4 nodes simultaneously for specific time points.

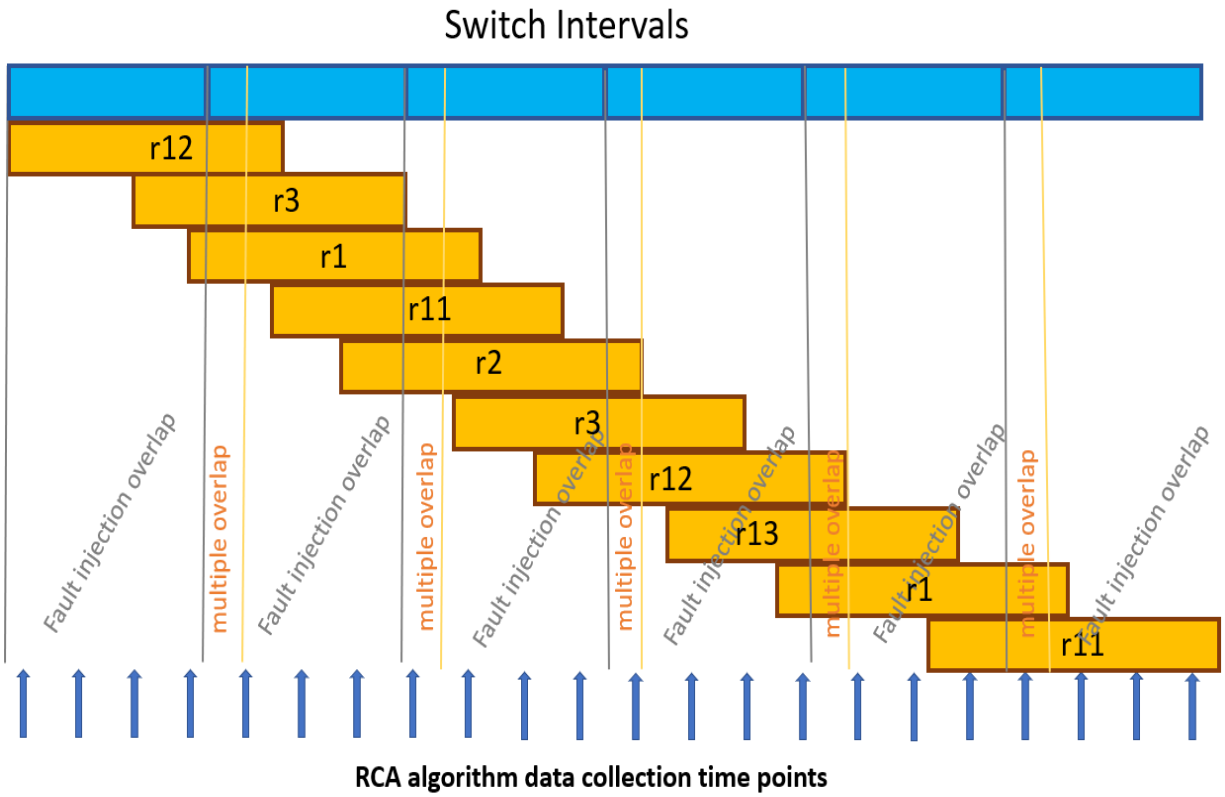

Figure 28: Scenario 4 - Time Evolution

The results of the experimental measurement of this 4th experimental scenario are showed in the below Figures.

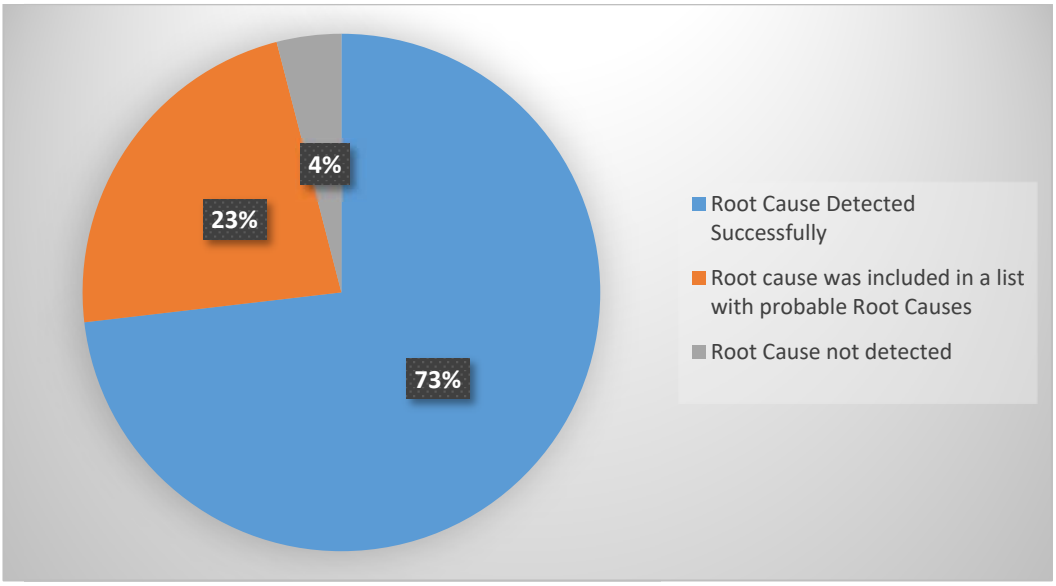

Figure 29: Scenario 4 - Data Collection Frequency 10 Sec.

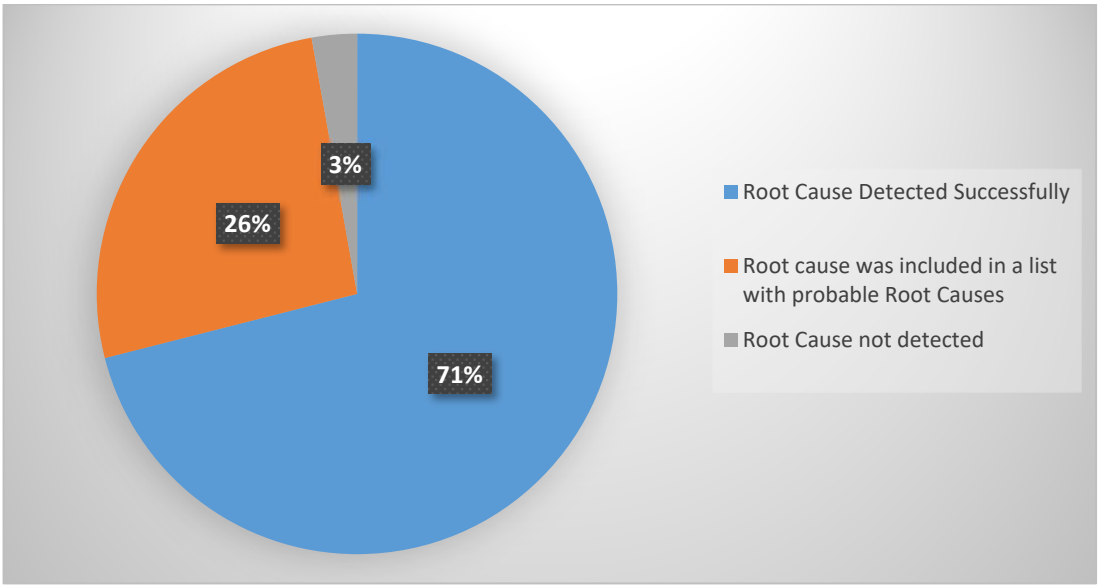

Figure 30: Scenario 4 - Data Collection Frequency 15 Sec.

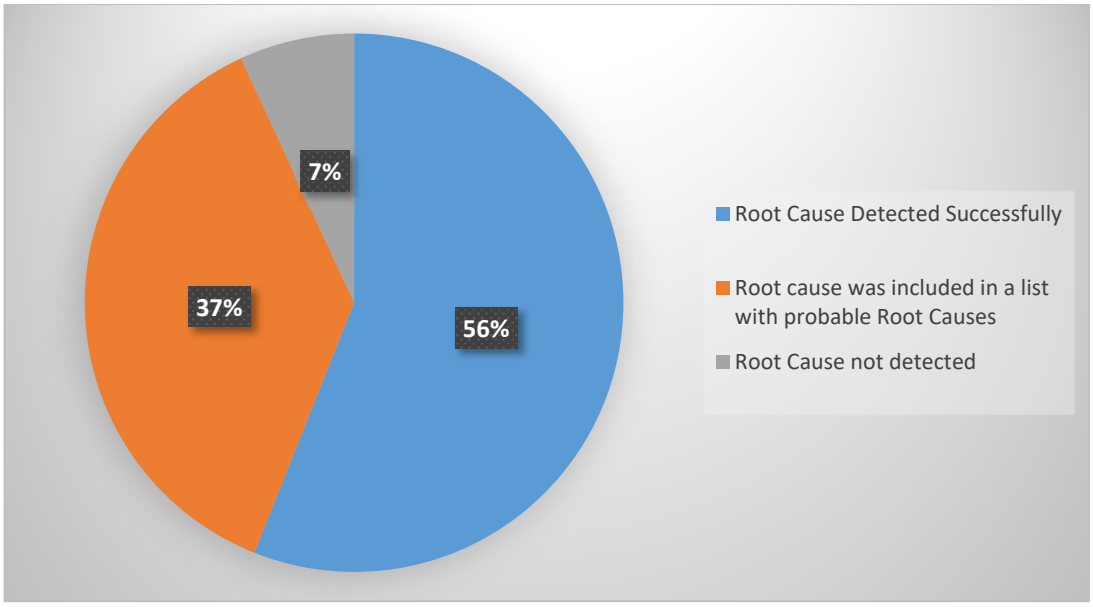

Figure 31: Scenario 4 - Data Collection Frequency 30 Sec.

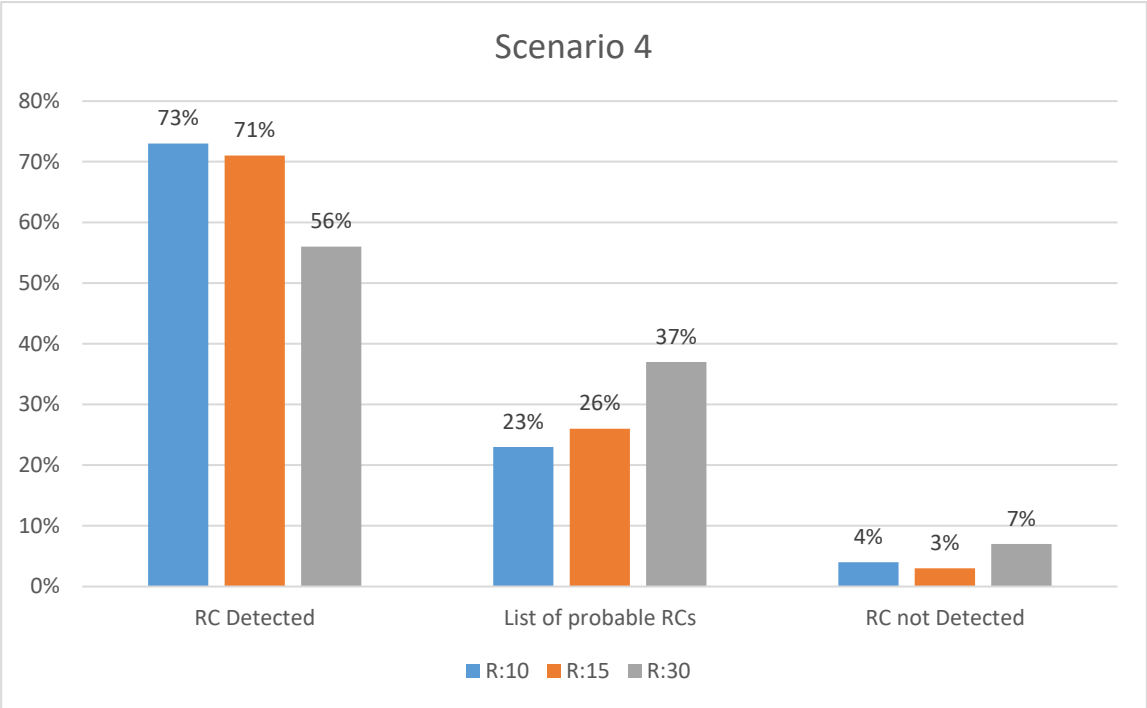

Figure 32: Scenario 4 overall results

As we can see, despite the fact that the we have increased the impairment time in comparison with the previous experimental scenario the algorithm still performs very well with the overall accuracy reached 96% when the data collection frequency is set to 10 seconds, 97% overall accuracy with data collection frequency set to 15 seconds and finally, 93% overall accuracy when data collection frequency is set to 30 seconds.

6.5 Scenario 5

In this scenario we will increase the Impairment time even more in order to check the RCA Algorithm's performance. The time variables have been set for this experimental scenario as Table 6 shows below.

Table 6: Scenario 5 Time Variables

Switch Time	Impairment Time	Algorithm data collection frequency
20 Sec.	140 Sec.	10, 15, 30 Sec.

Same as the previous scenario because of the timing configuration multiple overlaps are produced. Impairments are introduced at 5 or 6 nodes simultaneously for specific time points.

The results of the experimental Scenario 5 are demonstrated below

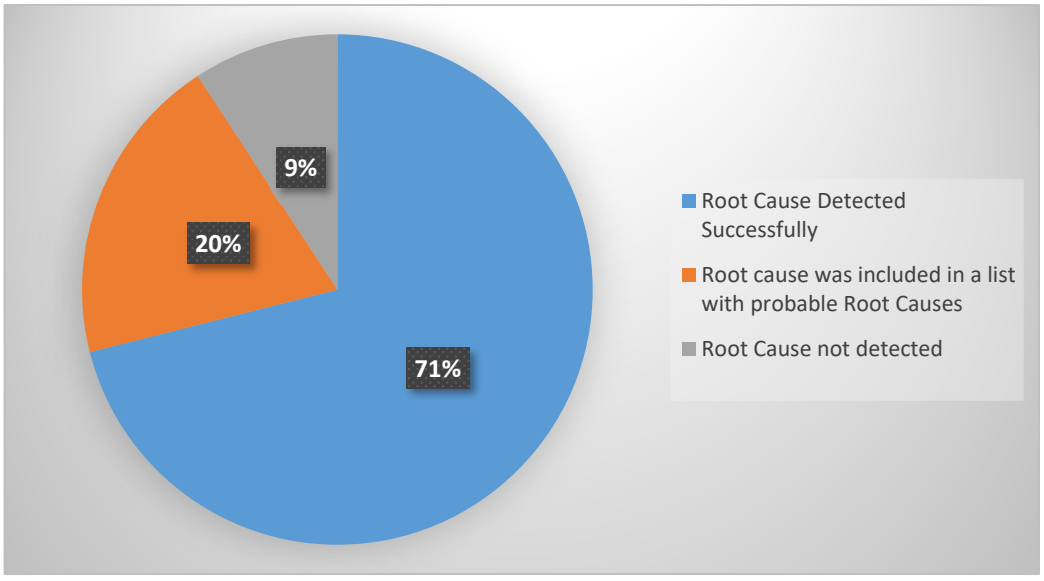

Figure 33: Scenario 5 - Data Collection Frequency 10 Sec.

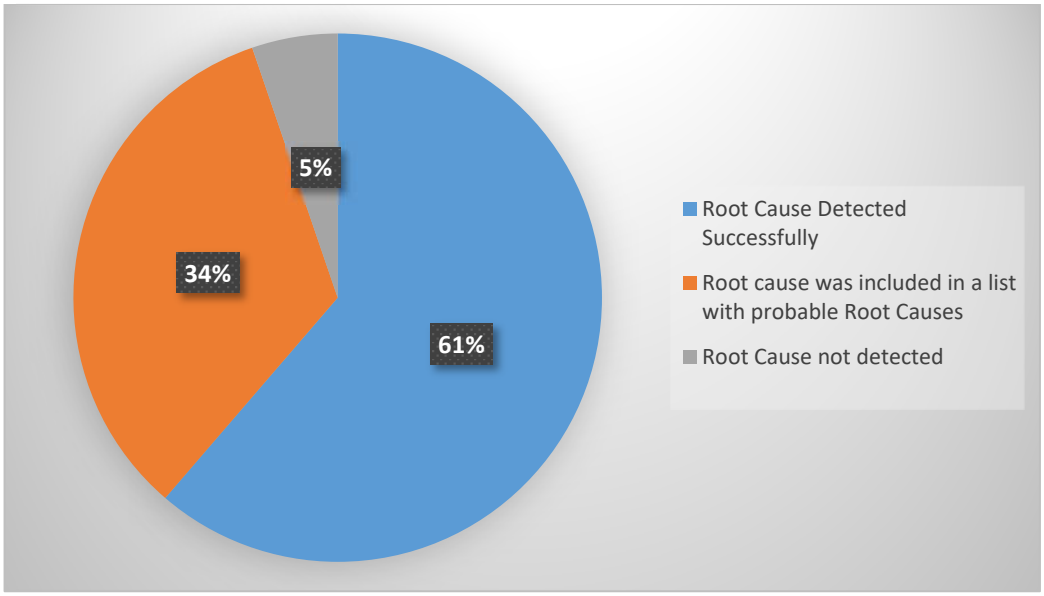

Figure 34: Scenario 5 - Data Collection Frequency 15 Sec.

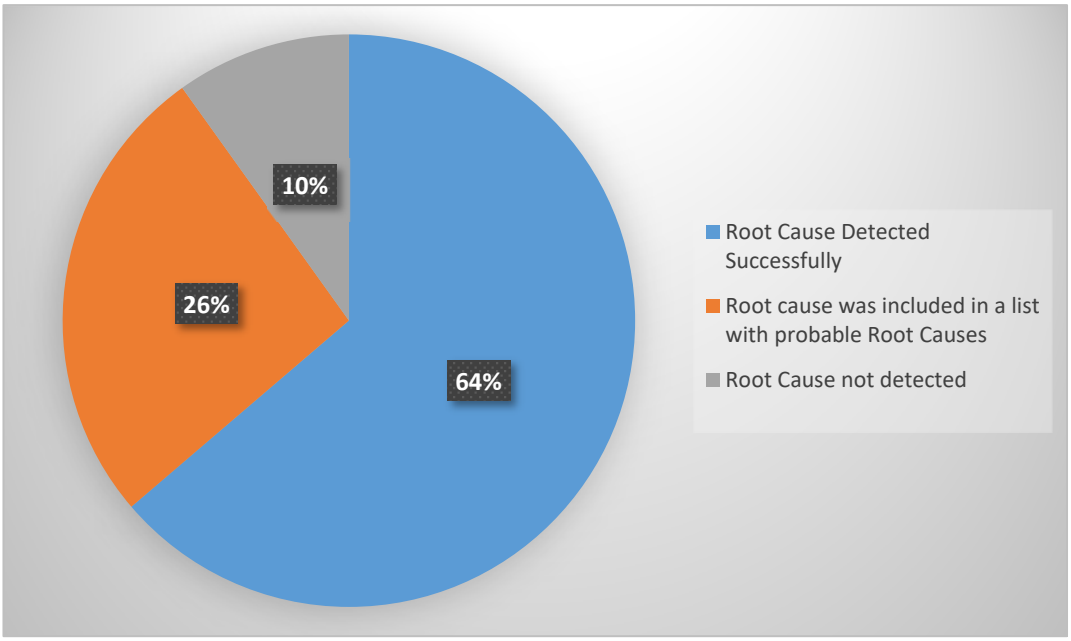

Figure 35: Scenario 5 - Data Collection Frequency 30 Sec.

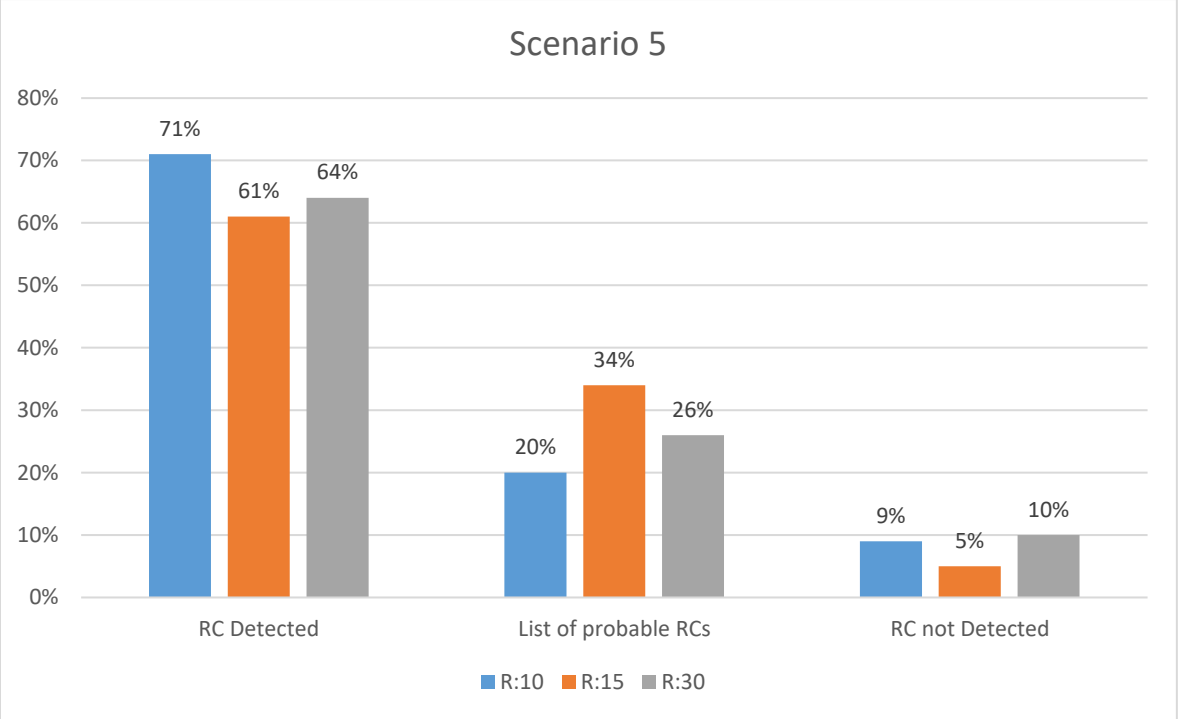

Figure 36: Scenario 5 overall results

Based on the above results we observe a small decrease in the overall accuracy in comparison with experimental scenario 4 but still the algorithm's performance remains high since the overall accuracy reached 91% when the data collection frequency is set to 10 seconds, 95% overall accuracy with data collection frequency set to 15 seconds (61% RC Detected and 34% List if probable RCs) and finally, 90% overall accuracy when data collection frequency is set to 30 seconds.

7. Conclusions

Through this work, a performance diagnosis platform for 5G services has been implemented and demonstrated. An RCA algorithm has developed and implemented to identify the cause of a system degradation and its evaluation shows the effectiveness of the RCA algorithm.

From the experimental scenarios we conducted:

- The RCA algorithm was successfully implemented in all the scenarios and was able to precisely detect the root cause in most of the cases.

- In all the scenarios, in cases when the exact root cause was not identified, the RCA algorithm provided a list of possible root causes in which the correct root cause was always included.

- When the Switch time is greater than the Impairment time (Scenario 1) the RCA algorithm is performing well, but the percentage of the cases that it could not detect the root cause is much higher

- In most of our experimental scenarios the RCA algorithm can detect exactly the root cause when it is fetching data every 10 seconds.

- Even though we have increased the Impairment time in order to produce multiple overlaps, the RCA algorithm's results are considerably successful.

REFERENCES

[1] Goyal P, Buttar AS. A study on 5G evolution and revolution. International Journal of Computer Networks and Applications (IJCNA). 2015 Mar;2(2).

[2] S, S.R.; Dragičević, T.; Siano, P.; Prabaharan, S.R.S. Future Generation 5G Wireless Networks for Smart Grid: A Comprehensive Review. Energies 2019, 12, 2140.

[3] Nancy Samaan, Ahmed Karmouch “Network Anomaly Diagnosis via Statistical Analysis and Evidential Reasoning” , IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 5, NO. 2, JUNE 2008

[4] Mfula, Harrison & Nurminen, Jukka. (2017). Adaptive Root Cause Analysis for Self-Healing in 5G Networks. 136-143. 10.1109/HPCS.2017.31.

[5] https://containernet.github.io/

[6] https://www.elastic.co/what-is/elk-stack

[7] https://kafka.apache.org/documentation/

[8] Javed Ahmed Shaheen, “Apache Kafka: Real Time Implementation with Kafka Architecture Review”

[9] Saurabh Chhajed, “Learning ELK Stack”

[10] Krishanu Bhattacharya, N.Usha Rani, Timothy A.Gonsalves, Hema A. Murthy, “An Efficient Algorithm for Root Cause Analysis”