

UNIVERSITY OF ZIMBAWE



Network Monitoring System Development: A Unified Multi-Vendor Network Monitoring System

BY

ARTWELL MAGADZIRE

R1713496

Submitted in partial fulfilment of the requirement for the degree of

MASTER OF SCIENCE IN COMMUNICATIONS ENGINEERING DEGREE

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING IN THE
FACULTY OF SCIENCE AND TECHNOLOGY

JULY 2020

SUPERVISOR: DR T MARISA

DECLARATION

I, Artwell Magadzire hereby declare that this dissertation for the Master of Science in Communication Engineering is my personal effort. Any other data or content I used as part of my research has been referenced and acknowledged

Name: Artwell Magadzire

Registration Number: R1713496

Signature:

Date: 16 July 2020

Supervisor: Dr T Marisa

Date:

ACKNOWLEDGEMENTS

I would like to express my gratitude and sincere thanks to my supervisor Dr. T Marisa for his continuous support and guidance on my research through which I developed critical and objective thinking and design skills and without whom, this final presentation would not have been possible.

Many thanks to the Faculty of Science and Technology in general and the Department of Electrical and Electronic Engineering lectures for the meticulous delivery of lectures and laboratory exercises

I would like to thank my fellow classmates and workmates Takawira Dzoro and Chance Kandishaya who helped me gather data, and offered guidance and assistance during this research paper. I also want to acknowledge the support received from the team at PolyNet Hungary for their assistance with access to their research and development material. Without you all, I would not have made it.

May the good Lord richly and abundantly bless you all.

ABSTRACT

Network monitoring is the first step in providing and guaranteeing delivery of mobile communication services of uncompromising quality. From enabling quick responsiveness to network failures, to allowing for predictive network capacity requirements projections, network monitoring systems have quite a wide range of important uses.

Most current network monitoring systems are challenged in not being able to monitor multi-vendor products. Those that are, come with prohibitive and punitive feature and license activation fees as well as vendor support fees.

Therefore a web-based multi-vendor network monitoring system was developed using open source software to undertake real-time monitoring of mobile network elements. Written in Python programming language, the system is capable of monitoring a number of different vendor equipment without the need for paying exorbitant support and feature activation license fees.

This research work is part of a larger project for a unified network monitoring solution that included SNMP based data collection mechanisms and graphical user interface development

Keywords: Network monitoring, simple network management protocol, Django Framework

TABLE OF CONTENTS

DECLARATION	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
ABBREVIATIONS	xii
CHAPTER ONE: INTRODUCTION	1
1.1 Introduction	1
1.2 Background	1
1.3 Problem Statement	2
1.4 Aim	3
1.5 Justification	3
1.6 Objectives	3
1.7 Research Question	4
1.8 Significance of Study	4
1.9 Limitations of the Research	4
CHAPTER TWO: LITERATURE REVIEW	6
2.1 Introduction to Network Management	6
2.2 Network Management	7
2.3 Network Management Systems	8
2.3.1 Client-Server Model	8
2.3.2 Browser-Server Model	9
2.4 Network Management System Design	10
2.5 Simple Network Management Protocol	11
2.5.1 The SNMP Manager	12
2.5.2 The SNMP Agents	12
2.5.3 Management Information Base (MIB)	12
2.5.4 MIB Objects	13
2.5.6 The SNMP Protocol	14
2.5.7 SNMP Message	15

2.6	Monitoring Mechanisms	15
2.6.1	Selecting the Information Monitored.....	16
2.6.1.1	Throughput capacity.....	16
2.6.1.2	Utilization rate.....	17
2.6.1.4	The Utilization of CPU and Memory	17
2.7	The MVC Architecture.....	18
2.7.1	Models.....	19
2.7.2	Views	19
2.7.3	Controllers.....	19
2.8	Django Framework.....	19
CHAPTER THREE: METHODOLOGY		22
3.1	Introduction.....	22
3.2	Design Methodology	22
3.3	Selecting between a Web browser based application and a desktop application	24
3.4	Development Tools and Technologies	25
3.4.1	Server Machine Specifications	26
3.4.1.1	Advantages of the Operating System:	26
3.4.1.2	Limitations of the hardware:	26
3.4.1.3	Software Packages used in the development.....	26
3.4.1.4	Linux Ubuntu 16.04	27
3.4.1.5	Python3.....	27
3.4.1.6	Django Framework 3.02.....	28
3.4.1.7	Celery 4.4.0	29
3.4.1.8	Redis.....	30
3.4.1.9	SQLite3	31
3.4.1.10	PyCharm.....	31
3.4.1.11	Net-SNMP 5.8.1	32
3.4.1.12	Snmp-cmds.....	33
3.4.2	Design sequence for Web based NMS application.....	33
CHAPTER FOUR: RESULTS AND ANALYSIS		35
4.1	Introduction.....	35
4.2	Network Monitoring Server Design.....	35
4.2.1	The system design architecture	35

4.2.2 High Level Design	36
4.2.3 Low Level Design.....	36
4.3 System Design Results	39
4.3.1 Device Enterprise ID Collection.....	40
4.3.2 Receiving and Parsing SNMP traps.....	45
4.3.3 System constraints during trap handling.....	50
4.3.4 Database Structure and SNMP Traps Storage	51
4.3.5 Retrieving SNMP attributes from a database and pass them to SNMP libraries/tools.....	53
4.3.6 Task Scheduling.....	54
4.3.7 Challenges faced in running scheduled tasks.....	56
4.3.8 Device Heartbeat Polling	57
4.3.9 Interface Status Polling	61
4.3.10 Interface Performance Collection	63
4.3.11 Collecting and Storing Performance Data.	65
4.3.11 Processing performance data	65
4.3.12 Presenting Performance Data.....	67
4.3.13 Network insight and analytics.....	70
4.3.14 Effects of unstable interfaces	71
4.3.15 System properties and performance.....	73
4.3.16 Consolidating the system	75
4.4 Results Analysis and Discussion	78
4.1 System Design Performance Evaluation.....	78
4.2 System Memory.....	80
4.3 System Dashboard	81
4.4 System Device Templates.....	83
4.5 Network Elements Interface Utilization Measurement.....	85
4.6 Interface Alarms Monitoring	86
4.7 Comparison of results with other related works	88
CHAPTER FIVE: CONCLUSION AND RECOMMENDATIONS	90
REFERENCES	92
APPENDICES.....	107

LIST OF TABLES

Table 1: MIB Objects	13
Table 2: Comparison of Web based and Desktop based application	25
Table 3: Tool and Software applications for each layer development.....	25
Table 4: Vendor Enterprise ID's	41
Table 5: Modules created for the process of handling traps	46
Table 6: Methods created to save and process traps.	46
Table 7: Tap Processing Keys.....	49
Table 8: Fig: List of Devices' SNMP Attributes	53
Table 9: Task Scheduling.....	54
Table 10: Performance information collection	57
Table 11: Processing performance data	65
Table 12: Performance Display Attributes.....	67
Table 13: Values collected for an interface GigabitEthernet0/0/21	68
Table 14: Interface stability calculations in an interval of 10 minutes	72

LIST OF FIGURES

Figure 1: Major functions associated with Network Management	7
Figure 2: The overall framework of the system	10
Figure 3: SMI structure showing MIB file organization	13
Figure 4: SNMP Protocol Stack.....	14
Figure 5: The SNMP message carried within the protocol layers	15
Figure 6: The MVT Framework.....	20
Figure 7: Research Methodology Approach	23
Figure 8: Host machine logical processors and cores.	26
Figure 9: Django 3.0.2 installation screenshot	28
Figure 10: Celery 4.4.0 installation screenshot.....	29
Figure 11: Redis installation screenshot	30
Figure 12: Sqlite3 installation screenshot	31
Figure 13: PyCharm Community Version	32
Figure 14: NetSNMP installation screenshot.....	33
Figure 15: snmp_cmds package installation screenshot	33
Figure 16: Software Design sequence for the system application	34
Figure 17: High Level System Design Architecture.....	36
Figure 18: Low Level System Design architecture	37
Figure 19: General Schema of Cacti	38
Figure 21: System Code for Enterprise ID Collection.....	41
Figure 22: Design Logic for Device Enterprise ID Collection	42
Figure 23: Vendor statistics on dashboard.....	43
Figure 24: A detailed table for vendor details	43
Figure 25: Panel for creating a vendor.....	44
Figure 26: Huawei Network Node Overview	44
Figure 27: Cisco Network Node overview	45
Figure 28: ZTE Network Node Overview	45
Figure 29: Trap Processing Flow diagram.....	48
Figure 30: Interface disconnection alarm displayed on the User Interface	50
Figure 31: Interface disconnection alarm cleared.....	50
Figure 32: Synchronous trap processing	50
Figure 33: Asynchronous processing of traps.....	51

Figure 34: Traps stored in the database	51
Figure 35: Database Structure.....	52
Figure 37: SNMP v2 details for a device.....	54
Figure 38: Celery Beat Schedule configuration.....	55
Figure 39: Task module functions	55
Figure 41: Celery beat schedule block diagram.....	56
Figure 42: Polling Functions Code Snippet.....	58
Figure 43: The process of polling the heartbeat of a device	59
Figure 44: Device with failed heartbeat.....	60
Figure 45: Device with a successful heartbeat.....	60
Figure 46: Device Listing	60
Figure 47: Interface Polling Status Flow Chart	61
Figure 49: Interface database table	62
Figure 50: Interface listing on the NMS	63
Figure 51: Performance data collection flow chart.....	64
Figure 52: Performance data processing overview	66
Figure 53: Code snippet of how the Performance Manager class is built.	66
Figure 54: Building blocks for of presenting performance data to the UI.....	67
Figure 55: Configuring the performance instance	69
Figure 56: Interface Utilization plotted with thresholds	70
Figure 57: Performance graph for an interface	70
Figure 58: Network Insight dashboard	73
Figure 59: Listing of high utilization interfaces	73
Figure 60: CPU information	74
Figure 61: System memory before starting the application	74
Figure 62: System memory when running the application	75
Figure 63: System monitoring when running the application.....	75
Figure 63: System functional requirements:	76
Figure 64: Integration points.....	77
Figure 65: System KPI Trend Analysis for Interface Performance Request.....	78
Figure 66: OpManager Server and Node CPU Utilization Capacity.....	79
Figure 67: Unified Network Monitoring System Memory Utilisation	80
Figure 68: OpManager System Performance.....	81
Figure 69: Unified Network Monitoring System Dashboard View.....	82

Figure 70: Unified Network Monitoring System Expanded Dashboard View.....	82
Figure 71: OpManager Graphical User Interface Dashboard View	83
Figure 72: Unified Network Monitoring System Vendor List View.....	84
Figure 73: OpManager Network Monitoring System Device Templates View	84
Figure 74: Unified Network Monitoring System Interface Utilisation on Cisco Switch	85
Figure 75: OpManager ZTE Switch Interface Utilisation	86
Figure 76: Device Monitoring Snapshot.....	87

ABBREVIATIONS

NMS	Network Management System
MNO	Mobile Network Operator
EMS	Element Management System
NE	Network Element
POTRAZ	Postal and Telecommunications Registration Authority of Zimbabwe
GUI	Graphical User Interface
CLI	Command Line Interface
TMN	Telecommunications Management Network
OS	Operating System
PDH	Pleisonchronous Digital Hierarchy
SDH	Synchronous Digital Hierarchy
OSS	Operations Support Systems
CORBA	Common Object Request Broker Architecture
CMP	Certificate Management Protocol
OSI	Open Systems Interconnection
IP	Internet Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
ICMP	Internet Control Message Protocol
SNMP	Simple Network Management Protocol
UDP	User Datagram Protocol
MIB	Management Information Base
OID	Object Identifier
ASN	Abstract Syntax Notation

CPU	Central Processing Unit
MVC	Model-View-Controller
DTL	Django Template Language
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
API	Application Programming Interface
UI	User Interface
OPEX	Operational Expenditure
CAPEX	Capital Expenditure
CRUD	Create Read Update Delete
DTL	Django Template Language
B/S	Browser Server
C/S	Client Server
ASN	Abstract Syntax Notation 1
SMI	Structure of Management Information
MTTD	Mean Time to Detect
MTTR	Mean Time to Resolve
KPI	Key Performance Indicator
SOC	Service Operations Centre
NOC	Network Operations Centre

CHAPTER ONE: INTRODUCTION

1.1 Introduction

The non-stop growth and ever-increasing need to provide new and improved services has given rise to complex telecommunications networks that have had a domino effect of placing a greater emphasis on Network Management solutions. Whether through mergers and acquisitions, new service introduction, or just organic network evolution, most – if not all – mobile network operators (MNOs) currently operate heterogeneous networks, made up of a diverse variety and range of network equipment (NE) manufactured and supplied by different NE vendors. Implementation of novel customer services often require network element configuration changes and updates on different vendor equipment, and traditionally, this has been achieved by configuring directly each NE device or using proprietary Element Management Systems (EMS) or Network Management Systems (NMS), via a vendor-specific Command Line Interface (CLI) or Graphical User Interface (GUI).

Network Management Systems (NMS) are applications that enable engineers to have Operation and Maintenance capability of network devices locally or remotely located using wireless or wired connectivity. The main tasks of Network Management Systems are:

1. Configuring network elements to provide services
2. Configuring and activating these services to the customers
3. Making sure the services and devices are working and operating according to the desired configuration and agreed Service Level Agreements
4. Monitoring and collection of alarms and performance information from the network elements

1.2 Background

In today's economic climate, mobile network operators need to be more competitive and agile than ever before just to survive in addition to delivering a healthy return on shareholder investment at an acceptable level of risk. To meet this need for development, continued growth and evolution, mobile networks have grown into multi-vendor equipment, multi-network operating systems, multi-topology, and mixed complex networks. So the daily management of maintenance work become increasingly more complex, tedious and demanding to the teams tasked with configurations, operations and maintenance of this diverse equipment.

Configuration and optimisation teams are faced with a diverse range of different CLIs and GUIs that they must use to configure and optimise multi-vendor services, increasing the likelihood of errors, the time to market (because of the need to ‘context switch’ between the different vendor environments), and also the cost of training,. An ultimate consequence of this is an increase in operational expenditure (OPEX) under staff (training) costs for the businesses. The inherent human need to record data about the ‘interface’ between different vendor equipment often causes users to start using , tools and other applications such spreadsheets to configure and reconcile system-wide network parameters. Such spreadsheet driven approach often leads to human error and decreases productivity.

The currently available Network Management Systems, come with license, support and management fees and payments tied to their use and activation of any other functionalities and features, if at all they are available.

1.3 Problem Statement

The majority of network management systems in current use by most mobile network service operators are vendor specific, meaning that they can only undertake operations and maintenance activities of the network elements from the same vendor that they are made from. On the contrary, most, if not all mobile network service providers, have networks that comprises of network equipment from a diverse range of equipment manufacturers and suppliers. In addition to this, the application and operation of features and functionalities on most of the current network management systems is licensed and requires regular payments for the mobile service providers to enjoy the use of them.

According to Postal and Telecommunications Registration Authority of Zimbabwe (POTRAZ’s) Abridged Postal and Telecommunications Sector Performance Report First Quarter 2020, the economic environment impacts the telecommunications sector through service demand and consumption levels, operating costs, investment and given the current inflationary pressures in the economy, operating cost containment will be even more crucial for operators to maintain profitability as the growth of operating costs poses a threat to operator viability.[19]

This current situation has thus created a challenge for mobile network service providers on how to adequately and effectively continue to manage and operate the different and diverse network equipment from the different vendors within their network, taking into consideration

the need to manage costs, the need to provide adequate supporting resources such as staff and training, whilst maintaining the key performance and quality indicators to within acceptable standards or regulatory standards. There is thus a need for a network monitoring solution that is capable of monitoring multi-vendor network elements, without the limitations of foreign currency denominated vendor support and license fees for full and diverse system features availability

1.4 Aim

The aim of this dissertation is to design a Network Monitoring System that is not vendor specific and can manage various transmission mobile network elements. The Network Monitoring System should be capable of monitoring multi-vendor network elements, notably ZTE, Huawei and Cisco devices, in addition to carrying out performance measurement and output service alarms of these devices on to a dashboard.

1.5 Justification

The current network management systems currently in use in most, if not all the mobile network operators are vendor specific, meaning that they can only monitor network elements designed and developed by the same vendor as the management system. Features and functionalities for management of other vendors, even though possible, are only activated on the payment of forex denominated licence and support fees, which in the current economic environment that our mobile operators are operating in, are proving very difficult to service and maintain. There is therefore a need in the local market for a cost effective unified network management system that can be used to monitor multi-vendor network devices. The system can assist in saving the mobile operators and the country at large, the high foreign currency, license and management fees, paid out to equipment vendors for their network management systems currently in use.

1.6 Objectives

The objectives of the project are to:

1. Design a cost effective network monitoring system for mobile network elements
2. Incorporate within the network monitoring system, functionalities and features that will enable remote monitoring of mobile network devices from different vendors

3. Design a graphical user interface for the network monitoring system that will provide a dashboard view of all the devices monitored and managed by the network monitoring system.

1.7 Research Question

1. What network monitoring system design can be used for monitoring mobile network elements in a multi-vendor environment?
2. What features and functionalities of a network monitoring system are critical to derive the most out a network monitoring system?
3. What impact does a user interface for a network monitoring system have on the mobile operators?

1.8 Significance of Study

Zimbabwe is considered to be having a fast growing mobile telecoms market. The rigorous demand for delivery of services of uncompromising quality in the highly competitive mobile telecommunications services industry implies there is just as much need for cost management initiatives as much as there is revenue growth initiatives to improve and transform the business investments into profitable revenue streams. Part of the cost management initiatives entail cutting down or even doing away altogether with vendor support fees and license fees for system feature and functionalities activations. This has become ever so important in the current challenging economic climate that the mobile operators and the country as a whole is going through, where foreign currency acquisition and management is now more important than ever before. The significance of this study is thus to not only be an effective technical monitoring system for a technical challenge, but to be a cost cutting initiative to the industry and the country as whole.

1.9 Limitations of the Research

Due to the time constraints, scope of the study was restricted to focus only on IP transmission switching units from the three largest suppliers of mobile IP transmission equipment in Zimbabwe, which could result in an oversight on certain mobile equipment's from other equipment suppliers or other mobile equipment's used in other domains like Core Network. Hence the IP transmission network equipment used for this research were from Cisco, Huawei and ZTE.

The other limitations were:

1. There was lack of enough funding for a more comprehensive design that could incorporate more diverse network nodes as issues of capacity on servers, database size and ability to procure a wide range of vendor equipment would then need to be also addressed.
2. Due to confidentiality clauses in the equipment manufacturers policies most were unwilling to provide information about their equipment or their operation without the availing of Non-Disclosure Agreements.
3. The research and design was limited to only those nodes predominantly used in the Transmission part of a mobile network topology, typically switches and routers and did not include Core Network nodes like base station controllers or mobile switching centres mainly due to time and the above mentioned two reasons .

For the research study to progress these limitations were mitigated by

1. Narrowing the scope of the research to focus primarily on a common set of network elements to monitor, which was IP transmission switches and routers so as to reduce the demand for more system's capacity requirements and the need for a wider range of vendor equipment to test with.
2. Making use of Open Source software's, applications and vendor equipment information like MIB files that are freely available from the Internet

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction to Network Management

A Telecommunications Management Network (TMN) is an infrastructure which provides interfaces for interconnection between various types of operation systems and/or telecommunications equipment to manage a diverse range of telecommunications network devices and services, with management information being exchanged through these interfaces [1]. This gives rise to the concept of network management or network monitoring which is the ability to have control of network devices either remotely or locally on site for purposes of carrying out various functions such as operation and maintenance, configuration management of performance management amongst desired functions. Network management is important for network operators in reducing their operating costs and differentiating themselves from their competitors. Also network management provides opportunities to introduce new services which will attract new customers and generate additional revenue.

The main goal of the network management is to detect and correct problems as they occur in the network. This ability to be always aware of the network's status and performance allows a basis by which network management and maintenance priorities can be established. The lack of good network visibility across all the nodes in the network makes it difficult, if not impossible to differentiate between failures that are service affecting and non-service affecting. In the same vein it is also difficult to plan for future events – such as network enhancements or planned engineering work – without information of how the network is performing currently.

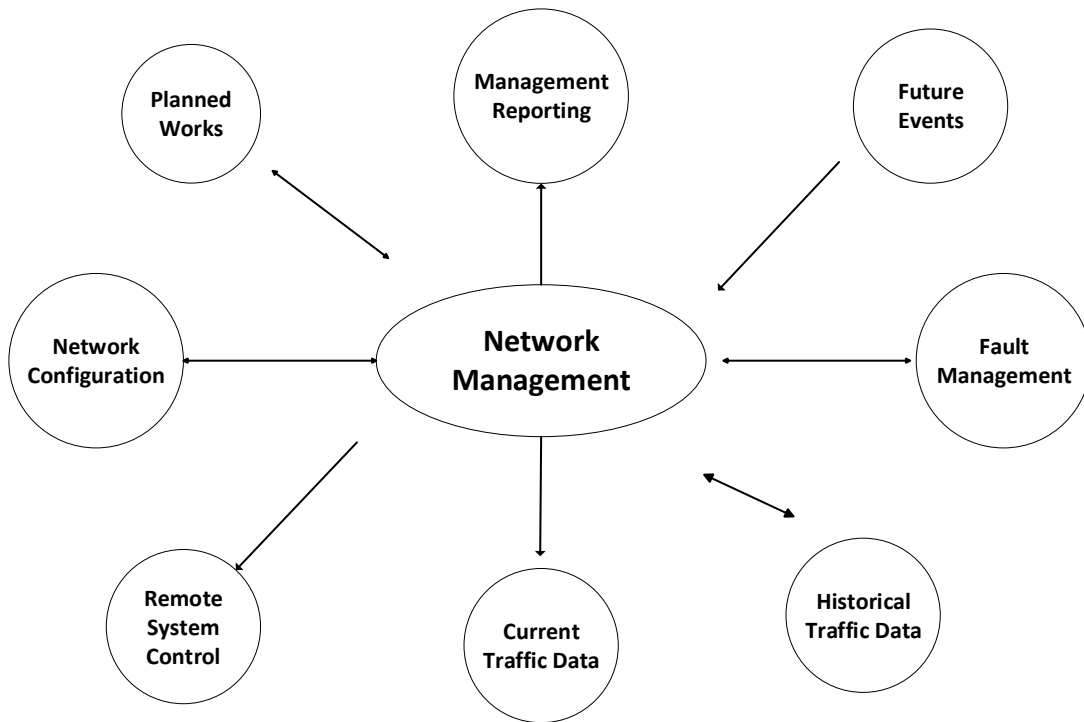


Figure 1: Major functions associated with Network Management [2]

As indicated in the diagram above some of the main functions of an NMS includes (but not limited to) the following:

1. Resource usage and hardware performance monitoring
2. To monitor the network round the clock without human intervention
3. Detecting operating system and application errors
4. To provide a view of the whole network traffic from one location.
5. Identifying computing assets by their name, location, or hardware characteristics.

2.2 Network Management

Traditionally provisions for network monitoring have always been included in network management systems. This is generally due the hierarchical arrangement that, to be able to manage a network, network monitoring needs to be possible. Thus network management requires network monitoring because monitoring is the core of network management that provides vital information about the network [3][15]. The distinction between network monitoring and network management systems is that management systems typically have more features that provide more efficiency to users.

The Simple Network Management Protocol (SNMP) [4] is an application layer protocol in network management. Though focussing on monitoring than management Astrolabe [5] is another management system. However there are systems like Supermon [6] and Ganglia [7] that focus exclusively on monitoring with the main user interaction being to view and analyse network information.

2.3 Network Management Systems

In the development of Network Management Systems, there are mainly two system models, that is, Client-Server model and Browser-Server model [8]. Alternatively as stated by V Geddes 2008 [15], most monitoring systems have a two-level architecture. The bottom level consist of monitoring daemons that are installed on network nodes, and collect and report useful information whilst at the second level in the architecture, is a hierarchical tree of aggregation nodes that pull data up from the agents, and make it available to clients in aggregated form [15]. C, C, Li. et al [16] opted for making Cacti use SNMP service to gather data from different network-attached devices such as routers, servers, switches etc in each and every interval determined by a cron poller

2.3.1 Client-Server Model

The Client-Server model is based on the distribution of application programs on either the client or the server, with the distribution layer and application logic hosted on the client and the data resource layer hosted on the server. The client completes certain tasks of calculation, through certain protocols and interfaces to communicate with the server, requesting the completion of service or obtaining data

Advantages of Client-Server Model

- i. Strong interaction: Development targeted, personalized customer interface design with intuitive, simple, convenient features, customized to meet customer operational requirements
- ii. More secure access mode: As the Client-Server mode is the point-to-point structural model, security can be better guaranteed.
- iii. Little traffic: Network traffic only between the client and server traffic
- iv. Fast Response: The client is directly connected with the server, without intermediate links

Disadvantages of Client-Server Model

- i. High Development Costs: Client-server structure of the client software requires higher hardware, in particular the continuing escalation of software .Higher requirements on the hardware increases overall system cost.
- ii. Difficult Transplanting: Compatibility between different platforms and different software makes developed tools difficult to transplant
- iii. Mixed user interface style, requires special training to use
- iv. Complex maintenance and difficult to upgrade.

2.3.2 Browser-Server Model

Based on Browser-Server mode three-tier architecture achieves the distribution of application layer, data resource layer and represent layer to different units. Represent layer is composed of the browser and dynamic web page , to receive and process the user's request and send to the Web application server. In the Web structure, the transaction layer and data logic layer are on the intermediate component, which is the key difference to the client-server structure. Intermediate component layer acts as a server, this is the Web application server. Application layer corresponds to the Web application server, business logic processing using the service of data resource layer to get the necessary information or to store, modify the corresponding data. Data resource layer corresponds to the database server to achieve the management of the database and data access, add, delete and update.

Advantages of Browser-Server Model

- i. Is not required on the operating system and software platform. Just installing the generic browser on the clients, the client can save hard disk space and memory and also the installation process is simple. Business expansion is made easy as to upgrade the system only requires upgrade on the server
- ii. Is especially suitable for online information to be published with unlimited number of front users. Users can expand arbitrarily with no need for additional investment in the long run, thereby greatly cutting costs

Disadvantages of Browser-Server Model

- i. Function weakening. Achieving special function requirements is difficult under the traditional model
- ii. Chances of achieving a personalised design are significantly lower.

- iii. The speed of Page dynamic refresh response is decreased [8]

2.4 Network Management System Design

Several Network Management Systems have been designed and implemented which include OpenNMS, Manage Engine, OpManager, Solarwinds, Zabbix, U2000, ZTE EMS, Ericsson's OSS , to mention but a few. With all these designs, the overall framework of system is as below

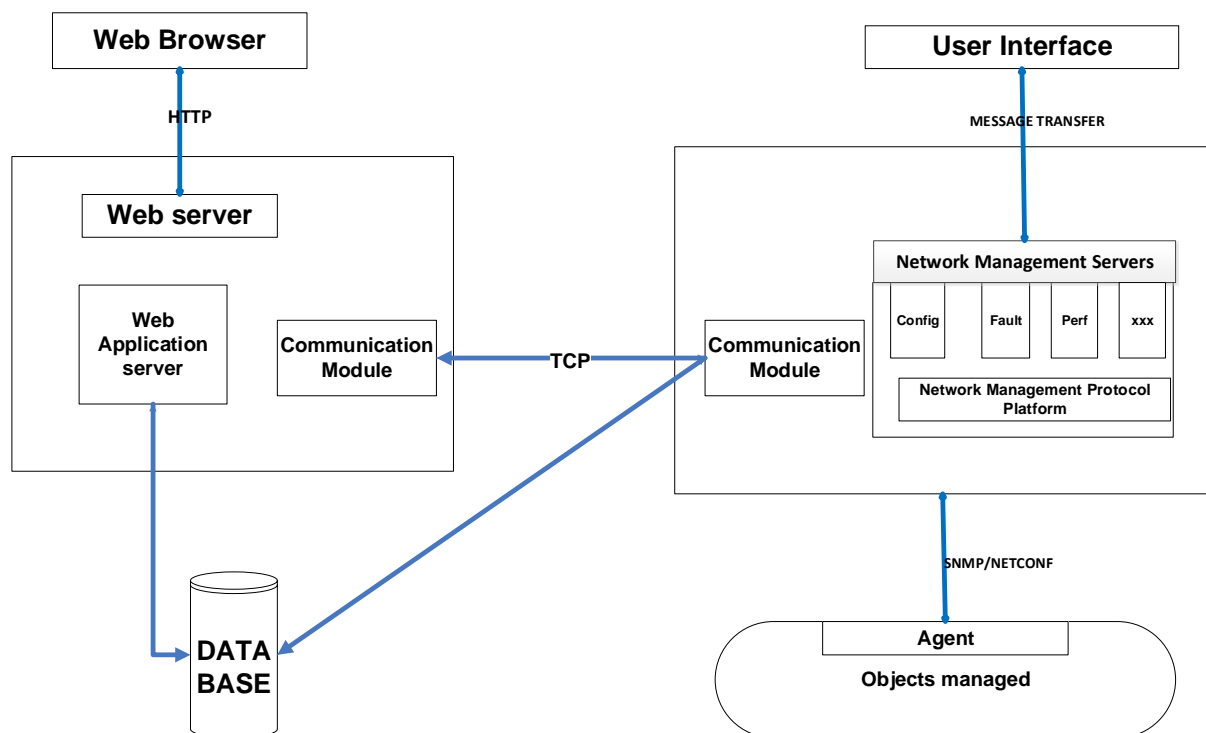


Figure 2: The overall framework of the system [8]

With this framework for network management system, the network management server is the core of the whole system management, and is composed of network management servers and network management protocol platform. Network management server modules act to provide a wide spectrum of comprehensive and effective management services for network and system, amongst which include services such as network topology, configuration management, performance management, fault detection and other traditional network management functions, as well as to also provide development interface for system services.

Network management protocol platform provides an interface for the network management server access to achieve communication with each Agent, to collect managed object management services information.

There are different ways of implementing Network Management Systems and these include, the Common Object Request Broker Architecture (CORBA) which is increasingly noticed as the base technology for the realization of higher layer functions of network management architecture. CORBA provides the framework for various object-oriented management applications to be able to function in a distributed environment. A feature of CORBA is that it guarantees access to the management information, independently of the software or hardware platforms. CORBA can support location transparency, and the integration of management information and services. Therefore, it could enhance the portability of applications that are developed across multiple network management platforms

Another option is to use CMP protocol management standards of OSI and the other is the use of SNMP management standards on the establishment of TCP/IP protocol. As TCP/IP has become the industry standard for networks, it has thus become the industry standard protocol widely used in various network management systems. Used widely in industry for carrying network management information, SNMP ensures that:

- i. Network elements management information can be transmitted between any two nodes.
- ii. Searching for, modification of, and location of equipment faults on any network device within the network is feasible and easily undertaken.
- iii. Fault diagnosis, capacity utilisation analysis, network upgrade projections, and various performance reports can be undertaken from a centralised point.

The features of the SNMP are as follows:

- i. To provide a basic function set through the use of polling mechanism. As such SNMP will fit, small-sized, quick, and low-cost network scenarios.
- ii. Makes use of the User Datagram Protocol (UDP) only.

2.5 Simple Network Management Protocol

The Simple Network Management Protocol (SNMP) is an application layer protocol used extensively in network management [4]. This protocol is a part of TCP/IP. Most of the modern tools support the SNMP. SNMP consists of the following components:

2.5.1 The SNMP Manager

The SNMP Manager is the one tasked with the responsibility to control one or more agents from a remote place. The Manager is software which is installed on the server. The manager can query SNMP agents by use of SNMP commands. When problems occur, the agent automatically informs the manager. The manager can also access the management information of the agent. The SNMP manager can also communicate to the network devices using commands to changing values in an agent's database. The SNMP manager provides the interface between the human network manager and the management system

2.5.2 The SNMP Agents

The SNMP Agents are software that is installed on the managed devices or the network devices to be monitored, either remotely or locally, allowing the agent to be able to communicate with the manager. The agent can record the management information about the network devices and give the response according to manager request. The main responsibility of the agent is to maintaining local management information and to give feedback to a manager through SNMP. All the management information are stored in its MIB. The SNMP agent can thus be said to provide the interface between the manager and the physical devices being managed or monitored.

2.5.3 Management Information Base (MIB)

The Management Information Base (MIB) is a collection of information which holds all the details about all network devices. Making use of this information, it is possible to identify a fault, isolate the fault and finally resolve the fault. Hence it provides useful information to monitor and manage the network operations. MIB is a part of SNMP agent software which keeps the information about objects such as variables. Each variable is allocated a unique identifier that is called an object identifier (OID) with information that is also accessible to the agnets. The SNMP manager will be able to access the MIB information for all the agents on the network.

2.5.4 MIB Objects

MIB objects define the following groups of objects in Table 1

Table 1: MIB Objects [2]

System	Name, location, description
Interfaces	Network interface statistics (traffic)
IP	IP statistics
ICMP	ICMP statistics
TCP	TCP statistics (TCP algorithms)
UDP	UDP statistics

Each SNMP managed device keeps a list of variables (objects), e.g. a router might have a variable called `buffer_overflow_count`. The exact format and name of these variables is standardized for a wide range of network devices. It is worth noting that the contents of the MIBs are defined by using Abstract Syntax Notation 1 (ASN.1). Each object can be represented using this language called Abstract Syntax Notation (ASN.1). The ASN.1 is a platform-independent language which allows for object (variable) definition [2]. Thus it is a language that allows a concise definition of the content in terms of numeric or alpha characters

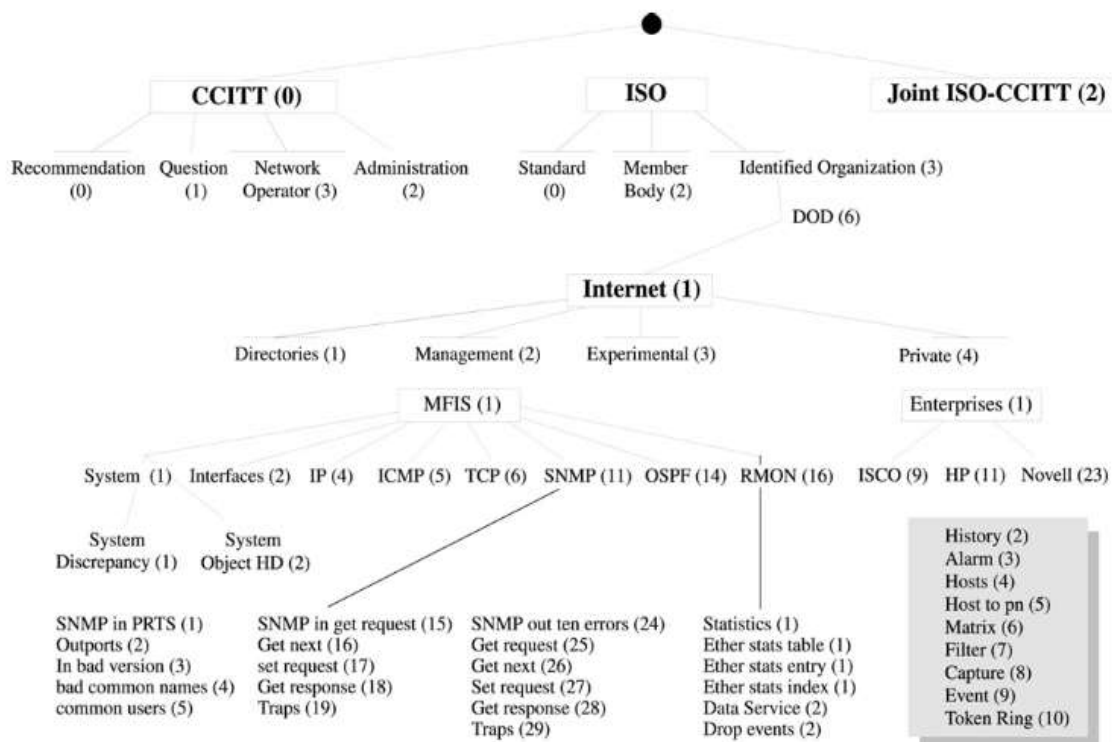


Figure 3: SMI structure showing MIB file organization [10]

MIB files are organized according to a logical structure, called a Structure of Management Information (SMI). The actual SMI structure is defined internationally by the RFC as a tree, using branches of the tree for various organizations. The diagram of the SMI structure above is a partially filled-in example of this tree. Each data element is unique because its path from the root through the various branches and twigs to the leaf is unique. Vendors may choose to customize the content of their MIBs under their internationally assigned vendor number [10]

2.5.6 The SNMP Protocol

The SNMP Protocol allows for the reading, writing and transferring of statistic information about network devices. Each message is transferred on internet using UDP. SNMP supports the TCP/IP protocol which is used to transfer the agent's queries and to make some changes in the objects. The snmp manager checks the state of the agent through periodical polls using UDP and IP protocol. The diagram below shows the architecture of the SNMP functions and how the functions are related to the ISO's Open Systems Interconnect (OSI) model.

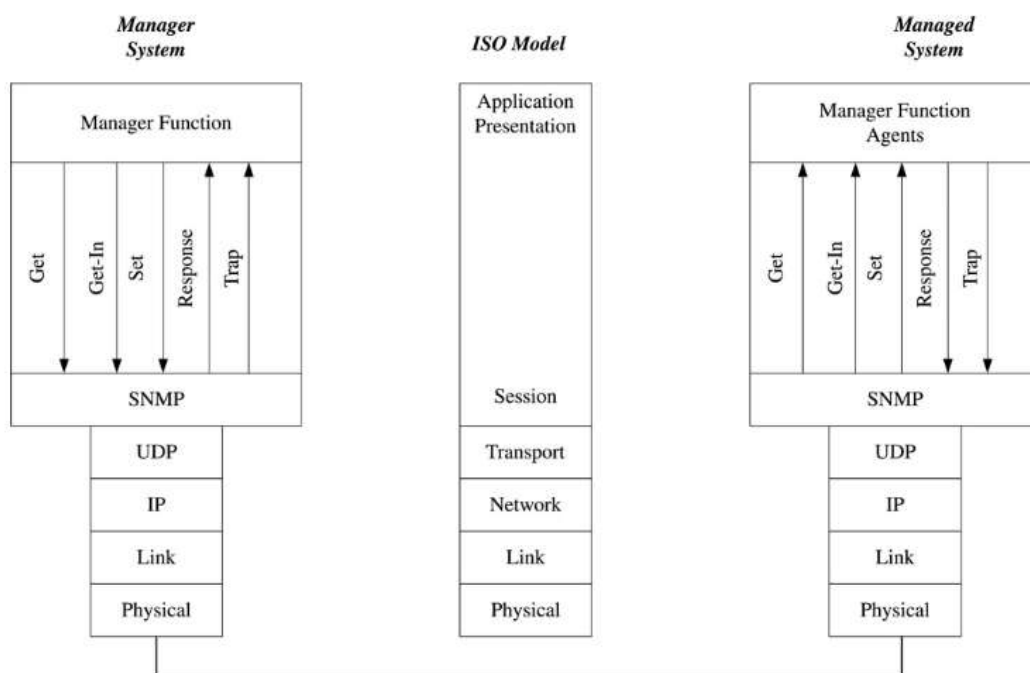


Figure 4: SNMP Protocol Stack

The SNMP protocol operates on top of the UDP protocol using port number 161 and 162 of UDP. For the agent the trap request message is port 161 whilst for the manager the trap messages port is 162. Because it works by exchanging a limited number of types of message.

The manager and agent can be communicating using three types of messages such as get, set and trap.

2.5.7 SNMP Message

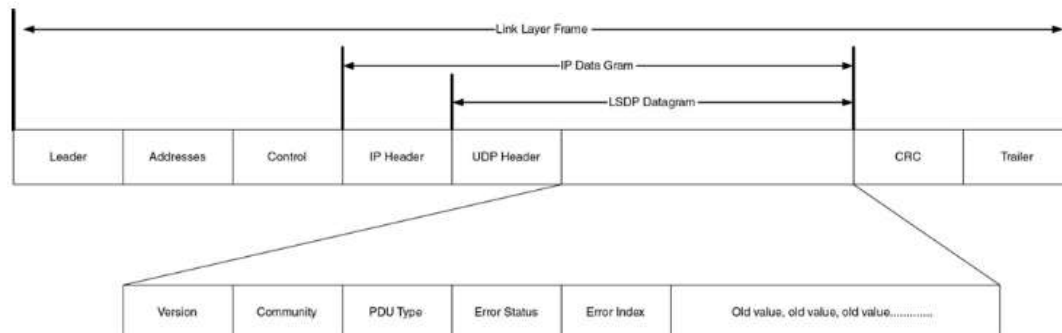


Figure 5: The SNMP message carried within the protocol layers

The diagram above shows how the SNMP message is carried whilst embedded within the protocol layers. Assuming we take it that the link layer protocol typically has some leader and header information and some trailer bits, the link layer frame carries the IP packet across the link layer connection to the next router. The IP datagram header has the routing information that lets the routers direct this packet to its final destination. Whilst the SNMP message itself is contained within the UDP datagram, the UDP header however does not carry much information.

The version field insures that the information exchange is with another agent of the same version. The community field is important because it is the security function. Thus the SNMP function can only collect information from members of a unique own community. This provides security for the network as it ensures that competitors or hackers will not be able to gain access to equipment attached to the Internet. The Protocol Data Unit (PDU) type specifies whether there is a GetRequest, GetNextRequest, SetRequest, Response, or a Trap. The error fields are used to identify SNMP errors, such as tooBig, noAccess, or badValue. The error pointer points to the location of the offending data field. Each Object Identifier (OID) then is included, followed by its value. [10]

2.6 Monitoring Mechanisms

Collection of information from the network equipment can be done in two techniques, roll poling and event report. Roll poling is a process that manages requests and responses between

the network manager and agent. Managers can query command and sent its agents within the scope of authorization and request of all kinds of information value, the agent will be the information in the MIB as a response. For real-time monitoring, managers must constantly go to polling agents and obtain data to evaluate the health of the network.

Event report is initiated by the agent, managers in a monitoring role wait to receive the information. Agent may be regular or pre-set cycle, is also possible when major events or abnormal events take the initiative to generate reports, which are very effective for real-time monitoring. For the state or value of relatively small changes in the monitored object can be more efficient than roll polling.

Roll polling and Event report is a network monitoring system adopted by the two kinds of the most effective methods, however, for different management systems, both have different emphases. Telecommunication management system uses more incident reports, and SNMP management does not depend on event report. The management of OSI system are more likely to find a balance point between the two ways, making the selection can be based on the following:

- i. Network data traffic generated by each method
- ii. Reliability of critical case
- iii. The required delay
- iv. Supporting the network monitoring application [9]

2.6.1 Selecting the Information Monitored

There are various key performance indicators that can be monitored for a Network Management System but the most commonly used indicator of most network monitoring systems is throughput, utilization and packet loss rate.

2.6.1.1 Throughput capacity

Throughput refers to the rate of data sent over the network that is an application-oriented indicators, usually represented as bits per second (bps), the number of bytes per second (Bps) or the number of packets per second (pps).

The formula is:

$$\text{Throughput} = \frac{\text{Bytes}}{\text{Time}}$$

(The number of bytes transmitted / an interval time)

2.6.1.2 Utilization rate

The utilization rate is the use of network resources to the frequency of the dynamic parameters, is more refined than the throughput indicator. It is used to search for potential network bottlenecks and congestion areas and also can know which resources have not been fully utilized. Through the analysis of network management information, it is possible to find that a resource is over used or the utilization rate is not high, adjust the network planning data, effect load balancing, and influence the effective use of resources

2.6.1.3 The interface utilization

Interface utilization is a key reflection of network utilization. The operational status of the interface can be tracked by monitoring the utilization rate, utilization rate can be expressed in percentage relative to the interface bandwidth as the number of bytes specified interface flowing. Calculations are performed using the collection of interfaces group variables, the formula is as follows:

$$\text{IfUtilizationRate} = \frac{\Delta \text{ifInOctets} + \Delta \text{ifOutOctets} \times 8 \times 100}{\Delta t \times \text{ifSpeed}}$$

Where:

Δt is time interval,

$\Delta \text{ifInOctets}$ is the number of the input byte collection,

$\Delta \text{ifOutOctets}$ is the number of output byte collection,

ifSpeed is the transmission rate of interface.

Similarly, you can use the interface table, where the number of input bytes and the number of output bytes are calculated input utilization and output utilization [9].

2.6.1.4 The Utilization of CPU and Memory

The CPU Utilization reflect equipment busy, and play an important role in the discovery of network congestion and balancing network load. However, The CPU Utilization is not defined in the public MIB but mostly in vendor-defined private variables. Cisco defines Cisco Process MIB (ID: 1.3.6.1.4.1.9.9.109) and Cisco Memory Pool MIB (ID: 1.3.6.1.4.1.9.9.48) to manage the CPU and memory [9].

A. Packet Loss Rate

Packet loss sometimes shows adverse signs of abnormal network, so packet loss rate is another important indicator of network monitoring. Usually less than 15% above this value network is usually not available. Acquisition interfaces group variable calculate, the formula is as follows: [4]

$$\text{DropRate} = \frac{\text{IfOutDiscard2} - \text{IfOutDiscard1} + \text{IfInDiscard2} - \text{ifInDiscard1}}{\text{Indrop} + \text{Outdrop}}$$

$\text{Indrop} = \text{ifInUcastPkts2} + \text{inInNUcastPkts2} - \text{ifInUcastPkts1} - \text{ifInNUcastPkts1}$

$\text{Outdrop} = \text{ifOutUcastPkts2} + \text{inOutNUcastPkts2} - \text{ifOutUcastPkts1} - \text{ifOutNUcastPkts1}$

DropRate is the packet drop rate;

IfInDiscard2 and ifInDiscard1 shows two different times the value of the ifInDiscard variable;

IfInDiscard shows the message input direction discard number;

ifInUcastPkts depicts the number of packets sent to the upper layer protocol subnet unicast communication;

ifOutUcastPkts shows the upper layer protocol packet number of requests sent to the address of the subnet unicast communication;

IfOutNUcastPkts shows the upper layer protocol and message request to non – unicast address subnet number [9,11].

2.7 The MVC Architecture

The MVC pattern was created to separate business logic from representation. The MVC architecture facilitates changes to the visual part and the business logic part of an app independently of each other, without any impact to the other.

With the current demands of a fast-paced, cut throat and competitive world, the use of web applications in the building of web-enabled applications has become the norm. Software frameworks used to establish web application, and website development, web services, and web resources are called Web application frameworks. A favourite type of web app framework is the Model-View-Controller (MVC) design separates the code for each application component into modules [12], [13], [14]. Using the MVC architecture an application is divided into the following three layers:

1. Model
2. View
3. Controller

2.7.1 Models

Models are a representation of how data is organized in the database. That is to say, in MVC pattern models define the database tables and structure as well as the relationships between these database tables.

2.7.2 Views

A view is what you see when you visit a site. For example, a blog post, a contact form etc., are all examples of views. A View is essentially a containment of all the information that is ultimately sent to the client i.e. a web browser, and generally, views are HTML documents.

2.7.3 Controllers

A Controller basically controls the flow and exchange of information. When a request for a page is made ,that request is passed to the controller to use programmed logic to decide on the nature of information that is needed to be pulled from the database and that which should be passed on to the view. The controller is basically the core of the MVC architecture because it acts as the interconnection between models and views.

2.8 Django Framework

Django is an open-source python web framework used for rapid development, pragmatic, maintainable, clean design, and secures websites. A web application framework is a whole set of all components needed for application development. The Django framework has the main advantage of allowing developers to focus on new components of the application instead of spending time on already developed components.

DJANGO ARCHITECTURE

Django follows the MVT framework for architecture.

- M stands for Model
- V stands for View
- T stands for Template

MVT is generally very similar to that of MVC which is a Model, View, and Controller. The work done by the controller part is in this case done by Django itself, and this is the main difference between MVC and MVT architectures. Django does this work of controller by making use of templates. Precisely, the template file is a mixture of HTML part and Django Template Language also known as DTL.

The three layers (Model, View, and Template) are responsible for different things and can be used independently.

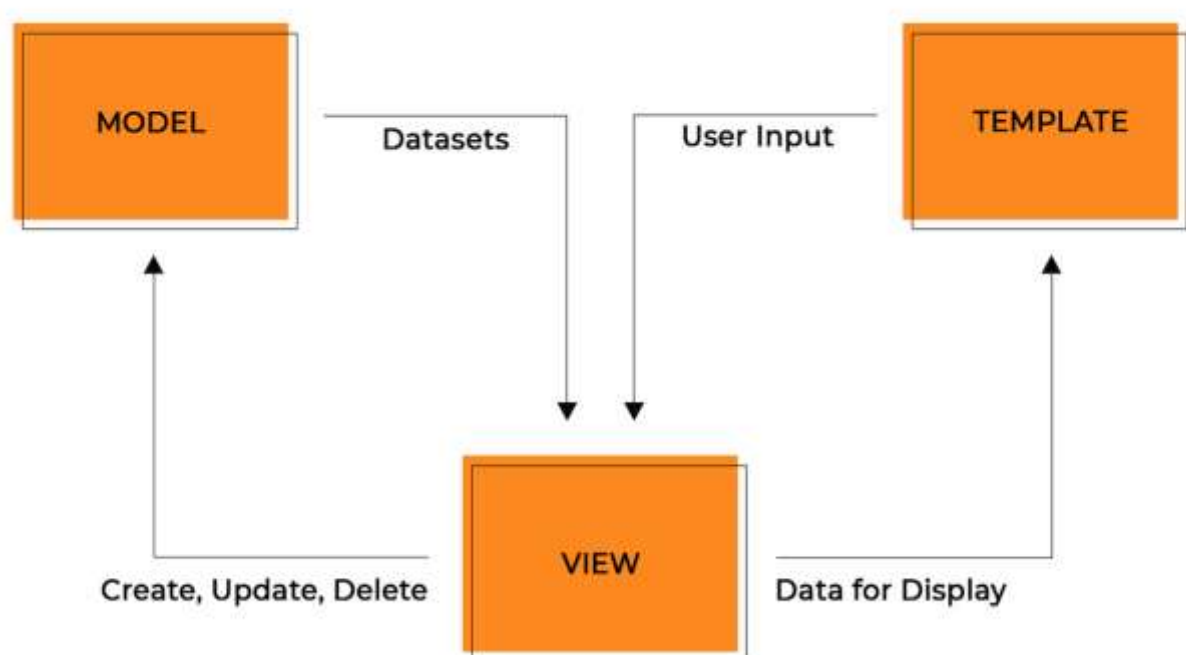


Figure 6: The MVT Framework

As the Django documentation states, a model is “the single, definitive source of information about your data. It contains the essential fields and behaviours of the data you’re storing.” Generally, each model maps to a single database table. Most applications currently available hardly operate without a database, and Django officially supports four: PostgreSQL, MySQL, SQLite, and Oracle.

Models contain information about your data and are represented by attributes (fields). A model is in essence a simple Python class, and as such it knows nothing about other Django layers. Communication between layers is possible only via an application programming interface (API).

Models contain all things related to data manipulation such as business logic, custom methods and properties, among a host of many. In addition, models enable developers to undertake actions such as create, read, update, and delete objects (data sets) from the original database.

The view's main tasks that it executes are: accepting HTTP requests, applying business logic provided by Python classes and methods, and providing HTTP responses to clients' requests. In other words, the view fetches data from a model and either gives each template access to specific data to be displayed or processes data beforehand.

Django has a powerful template engine and its own mark-up language with many tools such as templates, which are basically files with HTML code that is used to render data, the contents of which can be static or dynamic. Since there's no business logic in a template, it's there only to present data.

CHAPTER THREE: METHODOLOGY

3.1 Introduction

This chapter focuses at research methodology and experimental methodology. A research methodology is the specific procedures or techniques used to identify, select, process, and analyse information about a topic. It is also defined as the study of methods by which knowledge is gained or a framework that outlines the precise steps and activities for each phase and the output released from that phase before starting the next one.

This chapter goes into the details of how the design solution was developed, the development approach used and the factors that influenced the methodology and the design. Also detailed in this chapter are the software and hardware tools used in the development for the solution methodology. The descriptions give an overview of the software applications used, why they were chosen and their observed and known limitations. The chapter is organized in several sub-sections as follows:

1. Section 3.2 presents the detailed design methodology which was used to achieve the results and also the considerations made in the chosen attributes of the design.
2. Section 3.3 discuss the various tools and technologies that are used for this Network Monitoring System Development solution.

3.2 Design Methodology

In order to develop the network monitoring system, the strategy employed in this design process was to break down the design activities into three layers for easier and faster development.

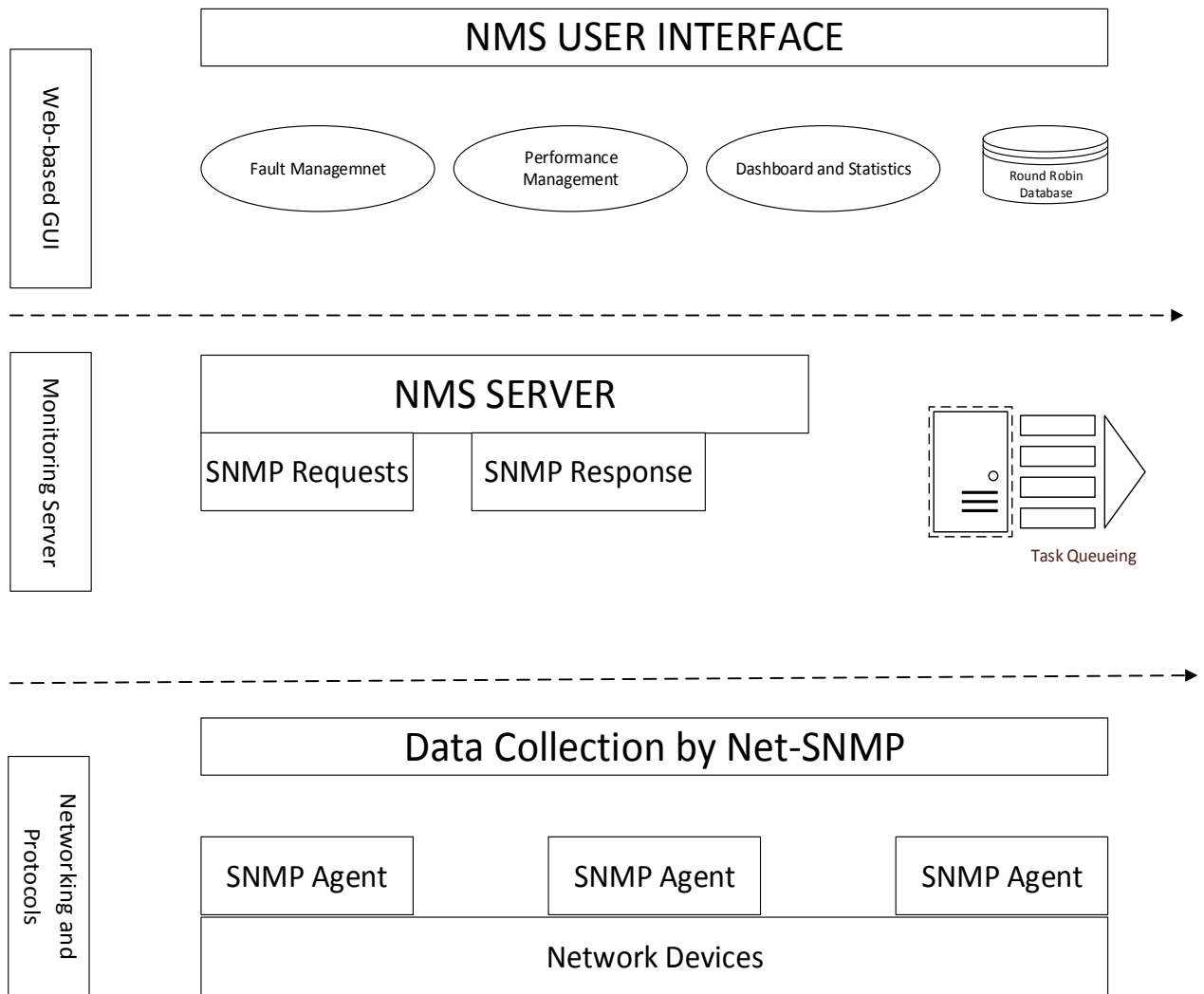


Figure 7: Research Methodology Approach

The system was designed around the principle of separation of concern. The main aim was for the system to be able to be designed in three distinct paradigms, so as to allow for continuous development without the whole system being shut down, as listed below

- Layer 1- physical interfacing with network devices, including protocols and mechanisms of information collection or delivery
- Layer 2- Monitoring Server Framework design concerned mainly with Data processing and control
- Layer 3- NMS user interface for input and output of the information with particular emphasis on data storage and presentation.

The strategy employed was to work on the three layers as independent parts and then integrate them all together once completed. This approach allowed the system to be scalable by being

able to modify each area without affecting the other parts of the system. The system could be modified without redeveloping the entire the system.

This document focussed mainly on layer 2- Monitoring Server Framework design, looking at the data processing and control paradigm of the system and also the integration points with the other paradigms. In designing the data and processing part of the network monitoring system the following main objectives were targeted:

1. Receive and parse SNMP traps using python
2. Retrieve SNMP attributes from a database and pass them to SNMP libraries/tools.
3. Generate SNMP requests and pass them to SNMP libraries/tools
4. Receive SNMP responses from SNMP libraries/tools and store the processed result to the database
5. Receive http requests and process them before sending the response
6. Carry out scheduled tasks
7. Create logs for troubleshooting system malfunction
8. Provide CRUD logic for devices and performance information.

3.3 Selecting between a Web browser based application and a desktop application

In the development of the solution a decision was made to have a web browser based user interface compared to native desktop application for clients. The table below shows the comparisons made before reaching the decision. For this system design, a web based application was the one used.

This is also the new trend in network management systems as they move away from traditional native desktop applications to web based Uis for clients. This approach removes the need to focus on the OS architecture of client devices but instead developers can focus on common browser specifications which are compatible across all devices.

Table 2: Comparison of Web based and Desktop based application

	Web Application	Desktop Application
Client device	Any device with a web browser	Client device should support the software.
Setup	Easy	Complex
Addition of features	Easy	Complex, may require to upgrade all client software each time.
Software Development	User interface is integrated in the server application	A separate software is developed for the server and the client
Connection	HTTP, HTTPS	HTTP,HTTPS
Operating System for the Server	Linux	Linux
Operating for clients	Any	Specific

3.4 Development Tools and Technologies

Several tools and software applications were acquired for the system design solution implementation. The table below shows the tools and software used for each of the 3 paradigms of the system.

Table 3: Tool and Software applications for each layer development

Paradigm	Tools
Networking and Protocols	NetSNMP
	SNMPCMDS
	Wireshark
Data processing and Control	Django Framework
	Celery
	Python3
Data storage and Presentation	Sqlite3
	Django templating System
	HTML, CSS and Bootstrap
	Redis

3.4.1 Server Machine Specifications

All software packages and applications used in the development of the Network Monitoring System were installed on the host machine running Linux Operating System which has the following specifications:

Processor: Intell CoreI i7-6700HQ CPU @1.90GHz 1.90 GHz

Installed Memory (RAM): 16.00 GB

System type: 64-bit Operating System, x64-based processor

The processor supports 2 physical cores for the execution of programs but it has 4 logical cores as shown under the CPU tab from the task manager in Figure 8 below.



Figure 8: Host machine logical processors and cores.

3.4.1.1 Advantages of the Operating System:

- It is Open source in nature and has easy installation and configuration requirements for software packages.
- It is widely supported by developers and it supports all the software packages required for this paper.

3.4.1.2 Limitations of the hardware:

Available RAM and number of processor cores could not allow for multiple simultaneous performance management jobs as the server's memory utilisation and processing power would decrease noticeably.

3.4.1.3 Software Packages used in the development

The following software packages were installed for use in the development of the Network Management System

- Linux Ubuntu 16.04
- Python 3
- Django Framework 3.02
- Celery 4.4.5

- Redis
- SQLite3
- Net-SNMP 5.8.1
- PyCharm
- Snmp cmds
- Widget Tweaks

3.4.1.4 Linux Ubuntu 16.04

Linux Ubuntu was used on the Server hosting the Network Monitoring System and is a modern, elegant and comfortable operating system which is both powerful and easy to use. Operating systems are programs that coordinate computer hardware and software. Software packages are developed for specific operating systems, therefore it is important to make sure software is compatible with the choice of operating system.

Advantages:

- i. Linux was the preferred Operating System chosen because it was both free of cost and is open source software.
- ii. It was easy to configure it for installing 3rd party software and other open source applications, therefore more time could be spent on the actual logic design.
- iii. In addition, Linux OS was chosen for its stability as compared to other Operating Systems like Windows because for Network Monitoring System with Availability as one of the key performance indicators of monitoring system, this was a key desired factor.

Disadvantages:

Some hardware drivers were not easily available for Linux. Most hardware manufacturing companies prefer to make drivers for windows or mac because they have more users as compared to Linux.

3.4.1.5 Python3

Python is a high-level programming language that incorporates dynamic semantics and is interpreted and object-oriented as well. It encourages program modularity and code reuse through the use of modules and packages. In addition the Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms

Deploying Python3 for the server

Python3 comes preinstalled in most distributions of Linux. Ubuntu 16.04 which comes with Python 3 preinstalled was used in the development of the system.

Why Python3 was used over Python 2

Python 3 was chosen for this project because Django 2.2 only supports Python 3.

Checking the version of python installed on our test machine

The command to confirm the version of python on the machine is shown below

```
python --version
```

3.4.1.6 Django Framework 3.02

Django is an open-source python web framework used for rapid development, pragmatic, maintainable, clean design, and secures websites. Being a web application framework, it is a toolkit of all components needed for application development. The main goal of the Django framework is to allow developers to focus on components of the application that are new instead of spending time on already developed components.



```
NMSENV) (base) artwell@artwell-HP-ZBook-15-G3:~/Projects$ pip3 show django
Name: Django
Version: 3.0.2
Summary: A high-level Python Web framework that encourages rapid development and clean, pragmatic design.
Home-page: https://www.djangoproject.com/
Author: Django Software Foundation
Author-email: foundation@djangoproject.com
License: BSD
Location: /home/artwell/Projects/NMSENV/lib/python3.6/site-packages
Requires: pytz, asgiref, sqlparse
Required-by:
```

Figure 9: Django 3.0.2 installation screenshot

Advantages:

- i. Django is one of the web frameworks which are written in Python programming language. Hence, it was easier to build the desired web applications with clean, readable, and maintainable code by taking advantage of syntax rules of Python
- ii. Django, like other modern web frameworks, supports model-view-controller (MVC) design rule. The MVC programming paradigm enable the keeping of the web

application's user interface (UI) and business logic layers separated. The approach enabled the reuse of the same business logic across multiple projects leading to the simplification and speeding up of the development of the web applications by separating their user interface and business logic layers.

- iii. Django also provided for easy customization, scalability, and ability to extend the web framework by making changes to its decoupled components. At the same time, the ORM system provided by Django allowed for common database operations and migration from one database to another without writing additional code.

Disadvantages:

Django, unlike other modern web framework, does not enable individual processes to handle multiple requests simultaneously as it lacks the capability to handle multiple requests simultaneously and a work around for this challenge was employed in the design using Celery

3.4.1.7 Celery 4.4.0

Celery is a Python based asynchronous task queuing software package that enables execution of computational workloads driven by information contained in messages that are produced in application code (Django in this case) destined for a Celery task queue. Celery can also be used to execute repeatable, periodic (i.e., scheduled), tasks



```
(NMSENV) (base) artwell@artwell-HP-ZBook-15-G3:~/Projects$ pip3 show celery
Name: celery
Version: 4.4.0
Summary: Distributed Task Queue.
Home-page: http://celeryproject.org
Author: Ask Solem
Author-email: auvipy@gmail.com
License: BSD
Location: /home/artwell/Projects/NMSENV/lib/python3.6/site-packages
Requires: pytz, billiard, vine, kombu
Required-by:
```

Figure 10: Celery 4.4.0 installation screenshot

Advantages:

- i. Celery allows for the execution units referred to as tasks to be executed concurrently on an single or multi server environment
- ii. Celery allows for tasks to execute asynchronously (in the background) or even synchronously (wait until ready)

3.4.1.8 Redis

Redis is an open-source, in-memory data structure store that is used as a database, cache, and message broker. In simple terms, it uses data structures like strings, hashes, lists, sets, bitmaps, and geospatial indexes to store data in the form of key-value pairs.

Specifically, Redis is used to store messages produced by the application code describing the work to be done in the Celery task queue. Redis also serves as storage of results coming off the celery queues which are then retrieved by consumers of the queue



```
MS2WV0 (base) [artur@tigerbait51 ~]$ ./redis-server
6379:C 02 Jul 2020 09:37:01.040 # Redis 6.0.10 (64 bit)
6379:C 02 Jul 2020 09:37:01.040 # Redis version=6.0.10, bits=64, commit=00000000, modified=0, path=/usr/local/bin
6379:C 02 Jul 2020 09:37:01.040 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
6379:~ 02 Jul 2020 09:37:01.051 # You requested maxclients of 10000 requiring at least 10032 max file descriptors.
6379:~ 02 Jul 2020 09:37:01.051 # Server can't set max open files to 10032 because of OS error: Operation not permitted.
6379:~ 02 Jul 2020 09:37:01.051 # Current max open files is 4096. maxclients has been reduced to 4094 to compensate for low ulimit. If you need higher maxclients increase 'ulimit -w'.

Redis 3.0-T (60000000) 64 bit
Running in standalone mode
Port: 6379
PID: 3135
http://redis.io

6379:~ 02 Jul 2020 09:37:01.053 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
6379:~ 02 Jul 2020 09:37:01.053 # Server initialized
6379:~ 02 Jul 2020 09:37:01.053 # WARNING overcommit_memory is set to 0! Background save may fail under low memory condition. To fix this issue add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
6379:~ 02 Jul 2020 09:37:01.053 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
6379:~ 02 Jul 2020 09:37:01.054 # Ready to accept connections
```

Figure 11: Redis installation screenshot

Advantages:

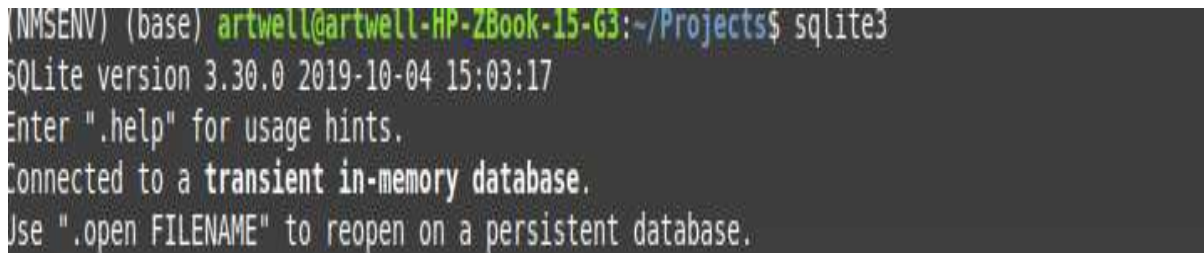
- i. Redis was chosen for being incredibly simple and straightforward to use, as well as being open source software.
- ii. By having all the data in memory, the latency issues on Redis have no comparison to other disk based DBs and requests can be processed at sub-millisecond latency
- iii. Redis has built-in geolocation storage capabilities, thus saving us the time of developing the logic

Disadvantages:

Because everything is in-memory with Redis, all data must fit in memory and so it's not possible to handle more data than you have memory and also lots of RAM is thus needed

3.4.1.9 SQLite3

SQLite is a software library that provides a relational database management system. It is a database engine that is self-contained, serverless, has zero-configuration and is transactional.



```
(NMSENV) (base) artwell@artwell-HP-ZBook-15-G3:~/Projects$ sqlite3
SQLite version 3.30.0 2019-10-04 15:03:17
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
```

Figure 12: Sqlite3 installation screenshot

Advantages:

- i. SQLite3 is about relational data. It can grow up into a very large database using standards. It's easier to build something and slowly migrate it upwards with SQLite based standards.
- ii. SQLite3 is very efficient at manipulating big datasets because of the SQLite index system
- iii. With SQLite3 it will be easier to upgrade for migration to a real relational database.

Disadvantages:

- i. As SQLite3 is a single-user DBMS in a multi-user system development as was this case in this one with multiple people working on the same database simultaneously, SQLite was found wanting, because it only allows single write at one time
- ii. Database size is restricted to 2GB in most cases.

3.4.1.10 PyCharm

PyCharm is an application that provides programmers and developers with basic tools to write and test software, specifically for Python programming. It is developed to operate across multiple platforms, including Windows, Mac OS, and Linux and consists of code analysis tools, debugger, testing tools along with version control options.

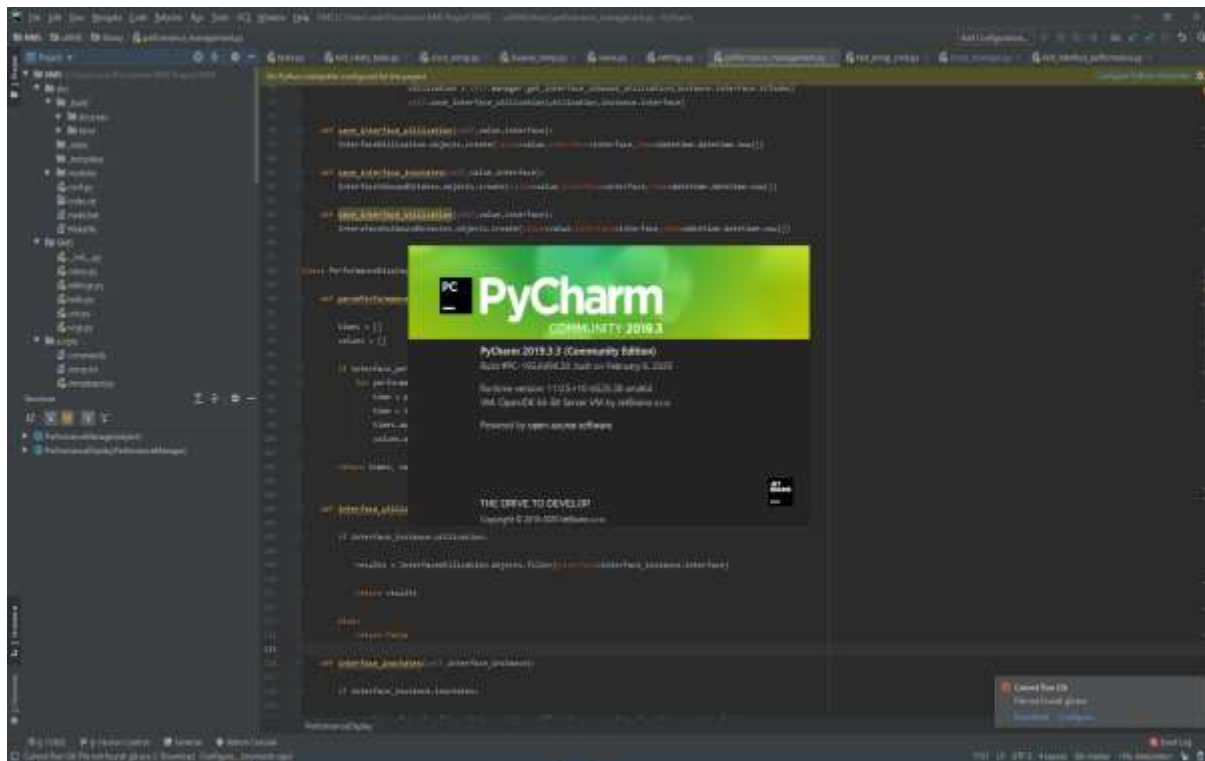


Figure 13: PyCharm Community Version

Advantages:

PyCharm provides a great suite of features for Python development, with the ability to run small code blocks separately without having to run the whole script, thus it helps to test blocks of codes separately and to debug.

Disadvantages:

- i. The high performance and multiple features that PyCharm offers come at a cost of high amounts of resource usage, particularly, the battery and the RAM, with at least 8GB RAM or 4 GB graphic card advisable for smooth functioning.
- ii. The enterprise version is quite costly.

3.4.1.11 Net-SNMP 5.8.1

Net-SNMP provides tools and libraries relating to the Simple Network Management Protocol including: an extensible agent, an SNMP library, tools to request or set information from SNMP agents, tools to generate and handle SNMP traps, etc.

```

NMSENV) (base) artwell@artwell-HP-ZBook-15-G3:~/Projects$ /etc/init.d/snmpd status
snmpd.service - Simple Network Management Protocol (SNMP) Daemon.
   Loaded: loaded (/lib/systemd/system/snmpd.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2020-07-02 11:28:13 CAT; 1h 46min left
     Process: 929 ExecStartPre=/bin/mkdir -p /var/run/agentx (code=exited, status=0/SUCCESS)
    Main PID: 931 (snmpd)
      Tasks: 1 (limit: 4915)
   CGroup: /system.slice/snmpd.service
           └─931 /usr/sbin/snmpd -Lsd -Lf /dev/null -u Debian-snmp -g Debian-snmp -I -smux mteTrigger mteTriggerConf -f

Jul 02 11:28:13 artwell-HP-ZBook-15-G3 systemd[1]: Starting Simple Network Management Protocol (SNMP) Daemon....
Jul 02 11:28:13 artwell-HP-ZBook-15-G3 systemd[1]: Started Simple Network Management Protocol (SNMP) Daemon..
Jul 02 11:28:13 artwell-HP-ZBook-15-G3 snmpd[931]: /etc/snmp/snmpd.conf: Line 145: Warning: Unknown token: defaultMonitors.
Jul 02 11:28:13 artwell-HP-ZBook-15-G3 snmpd[931]: /etc/snmp/snmpd.conf: Line 147: Warning: Unknown token: linkUpDownNotifications.
Jul 02 11:28:13 artwell-HP-ZBook-15-G3 snmpd[931]: Turning on AgentX master support.
Jul 02 11:28:14 artwell-HP-ZBook-15-G3 snmpd[931]: NET-SNMP version 5.7.3

```

Figure 14: NetSNMP installation screenshot

3.4.1.12 Snmp-cmds

Snmp-cmds is a python library for communicating with a target device through SNMP

```

NMSENV) (base) artwell@artwell-HP-ZBook-15-G3:~/Projects$ pip3 show snmp_cmds
Name: snmp_cmds
Version: 1.0
Summary: A python wrapper around the Net-SNMP command line utilities
Home-page: https://github.com/alextrenblay/snmp_cmds
Author: Alex Tremblay
Author-email: alextrenblay@github.com
License: MIT
Location: /home/artwell/Projects/NMSENV/lib/python3.6/site-packages
Requires:
Required-by:

```

Figure 15: snmp_cmds package installation screenshot

3.4.13 Design sequence for Web based NMS application

The actual process of designing and code writing and implementation followed a structured format as indicated in the diagram Figure 16. Drawing on lessons learnt from the shortcomings of the traditional vendor specific network monitoring systems in industry, a network monitoring system based on Django network framework was proposed and designed.

The first step was the setting up of the Django web framework, this created the base of the entire project. The setting up of Django includes the installation of the package, creating the project and the applications, setting up the database, configurations for Celery, integration with NetSNMP and finally the creation of deployment scripts.

The majority of the work was during the integration with NetSNMP where data was to be processed between the web application and the SNMP protocol tools. Due diligence was taken in order to accommodate SNMP data from multiple vendors and also to accommodate future expansions.



Figure 16: Software Design sequence for the system application

CHAPTER FOUR: RESULTS AND ANALYSIS

4.1 Introduction

This chapter discusses and presents the results of the research focussing mainly on the detailed design of the network management system and how the different components interact with each other and it also elaborates on the theory and flow charts behind the design.

Section 4.2 details the actual design solution of the network monitoring system, its architecture and hardware and software setup and connectivity.

Section 4.3 discuss how the different and various objectives of the main focus of this paper, which was Monitoring Server Framework design in particular, was achieved with the aid of process flow diagrams and code snippets of how a solution for each objective was achieved In addition results obtained or outcomes of from each solution to an objective on the Network Monitoring System are presented

Section 4.4 analyses and discusses the results of the designed solution in comparison with some current industry network management systems currently being used by the mobile operators in the country.

4.2 Network Monitoring Server Design

4.2.1 The system design architecture

As the system was designed around the principle of separation of concern, the high level and low level system designs were developed with this consideration, to allow for continuous and parallel development and improvements to the system, without necessarily having the whole system down.

The system is a web-server online monitoring system developed by the Django network framework based on the B/S architecture. The Django network framework follows the model-view-controller (MVC) control mode, which defines the code and the method of data access (model) separating from the request logic (controller) and the user interface (view) so that each component can be designed separately without affecting other components [17][18]. The model section is used to describe the data tables that were collected and interacted with the database in the server for data. The view part is the graphical user interface, that interface which the user sees and interacts with, displays the collected snmp data to the user, and receives the user's input data [11]. Simultaneously, the view also accepts data update events from the model, so

that our network element monitoring data can be displayed in the user interface in real time. The controller is responsible for logic processing, snmp requests, time, equipment and other data displayed on the view, calling the model to process the business request.

4.2.2 High Level Design

The high level system design shows that network management is hosted on a Django webserver and MySQL database, with a connection to the various network elements being monitored via SNMP. Several clients can be connected to the server via http for user interface connection.

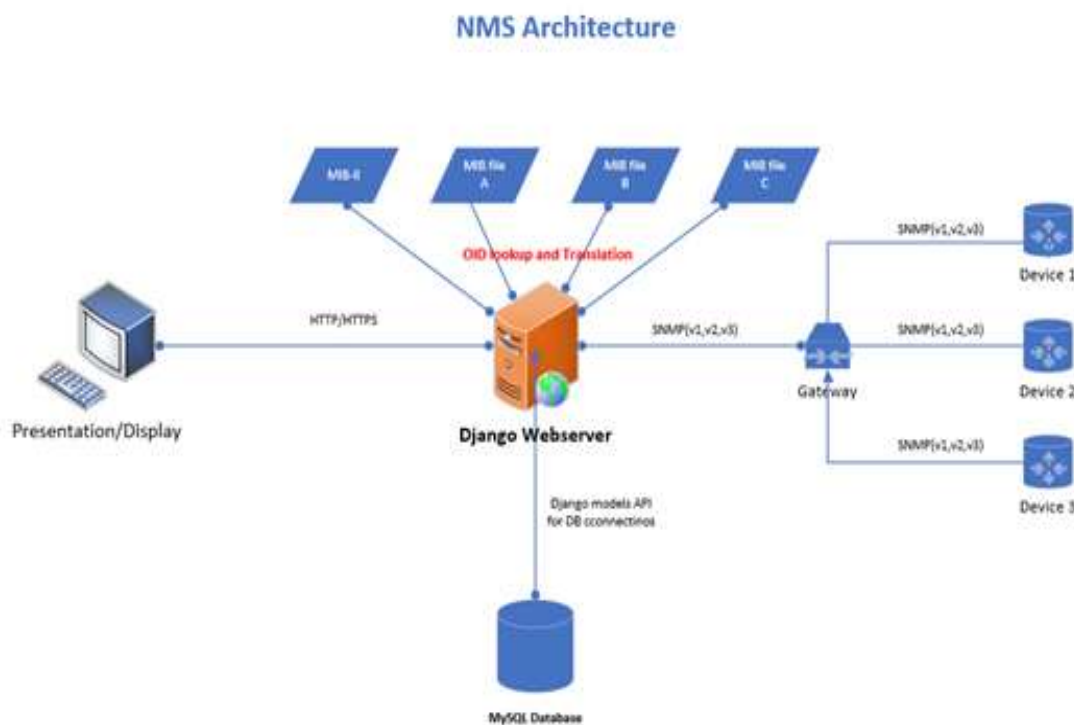


Figure 17: High Level System Design Architecture

4.2.3 Low Level Design

The low level design shows the three layers that were used in the system development which are

1. Networking, Protocols and device interfacing
2. Data processing and control
3. Data storage and presentation

The applications, interfaces, protocols and databases used for the system development are indicated in greater detail. Sqlite and Redis are the databases that were used in the system development as indicated in the diagram Figure 18.

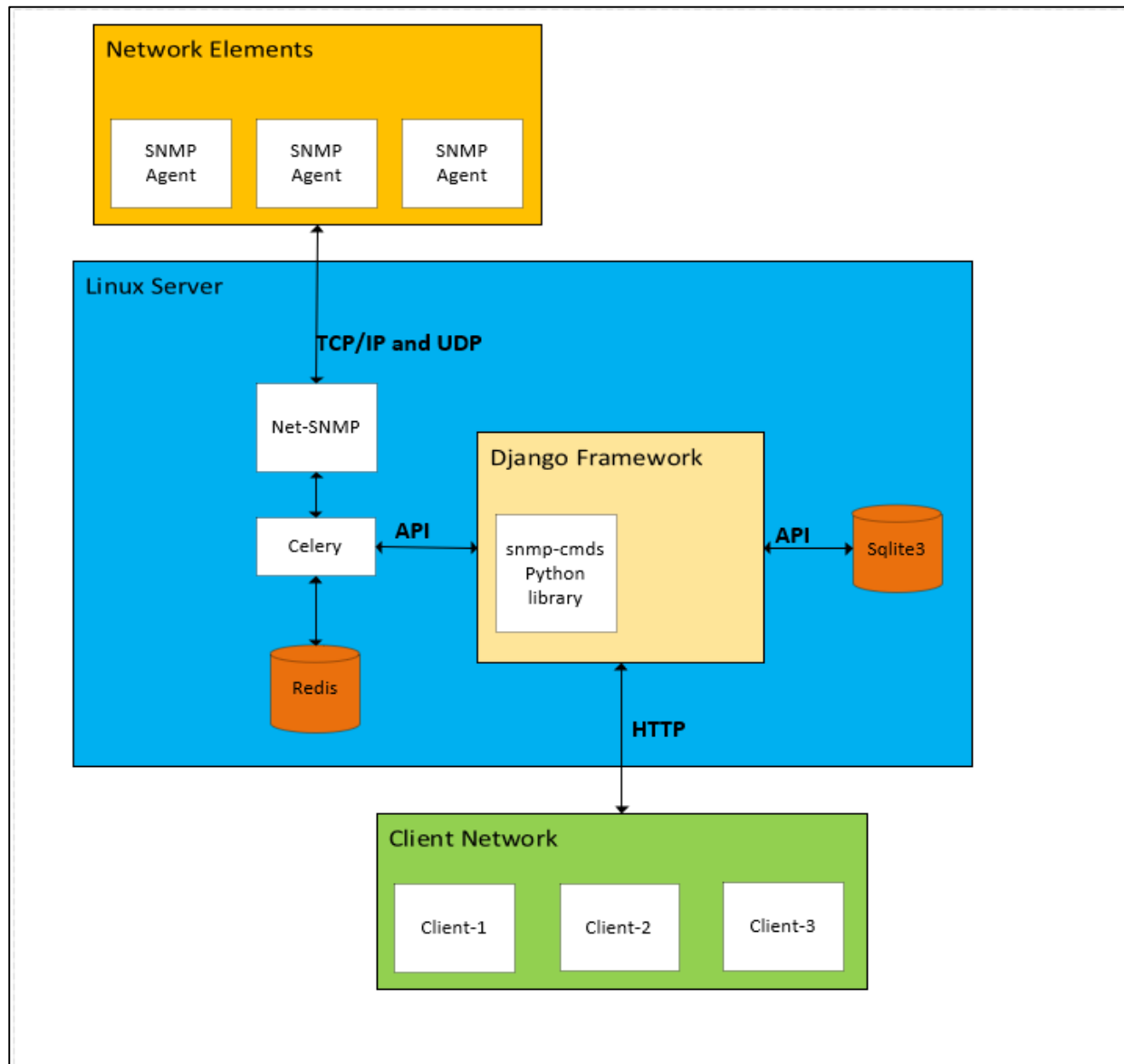


Figure 18: Low Level System Design architecture

The high and low level system design architecture has some similarities with that used in the design of the network monitoring system based on CACTI for the Experimental Advanced Superconducting Tokamak (EAST) [16].

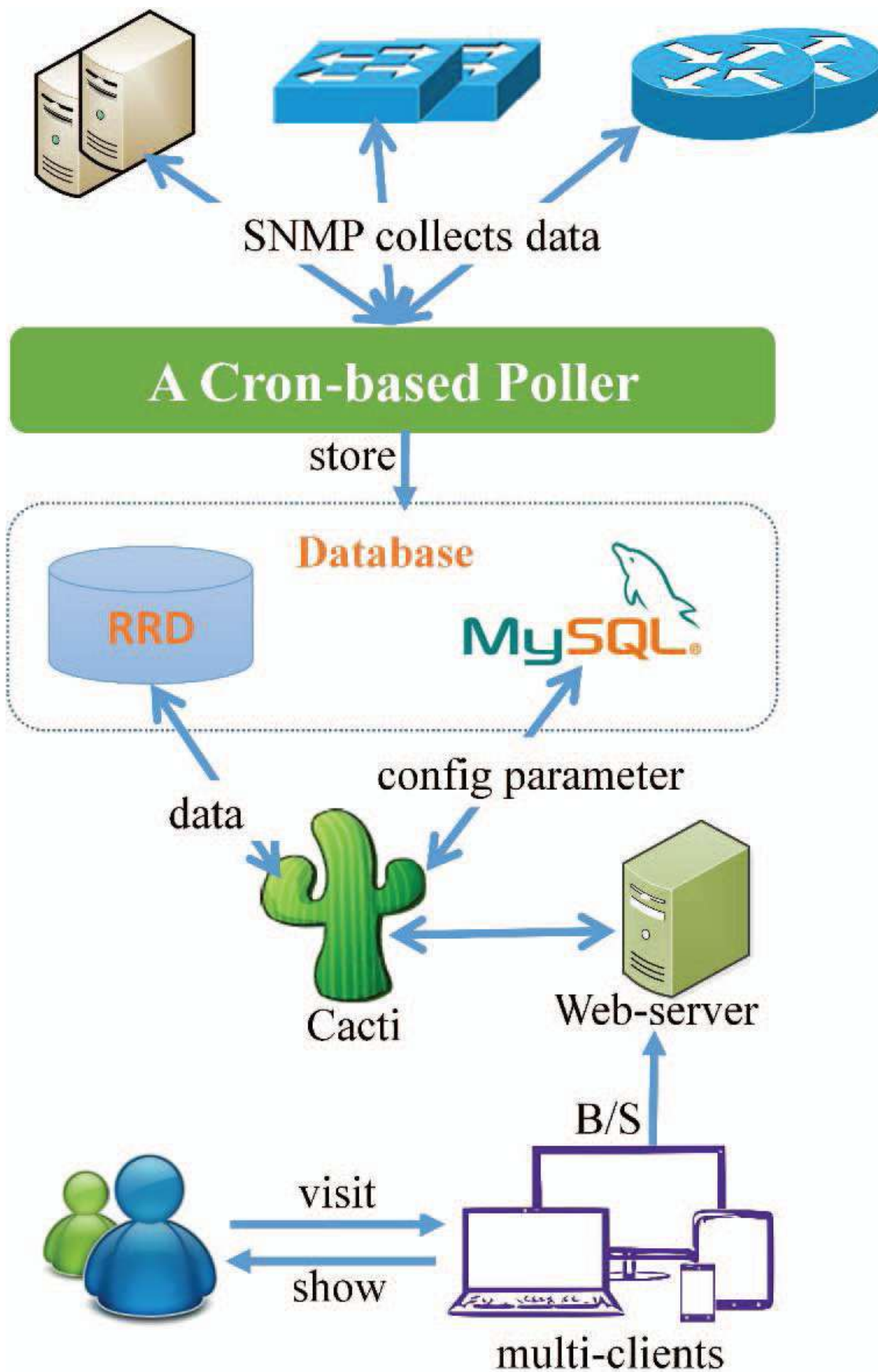


Figure 19: General Schema of CACTI [16]

Like the architectural design solution for this research, Cacti uses SNMP protocol to collect data from different network-attached devices such as switches, routers and servers, in each interval decided by a cron-based poller [16]. In addition, Cacti also makes use of two databases, Round Robin Database and MySQL database to store the polled and relevant configuration information.

4.3 System Design Results

In order to manage devices on the NMS a step of creating the devices was created first. The process of adding a device to the system provides the communication parameters to the management system prior to any communication between the NMS and the devices. The requirements for adding a device to the NMS were established as follows.

Information required before adding a device

- Device IP address
- SNMP protocol version
- Community strings
- Credentials (if using SNMPv3)
- Layer 3 reachability of the device from the NMS.

The process was meant to be vendor and model antagonistic. The system is supposed to determine the vendor and model via SNMP request to the device. The system design did not include the verification of the SNMP information during the creation of the device in the NMS, this feature can be added in further development of the NMS software.

The adding of devices on to the NMS is done in two steps

Step 1: create device by adding the IP address, location and device type

Step 2: Specify device SNMP credentials

All this information is stored in separate database tables.

Device creation and information collection

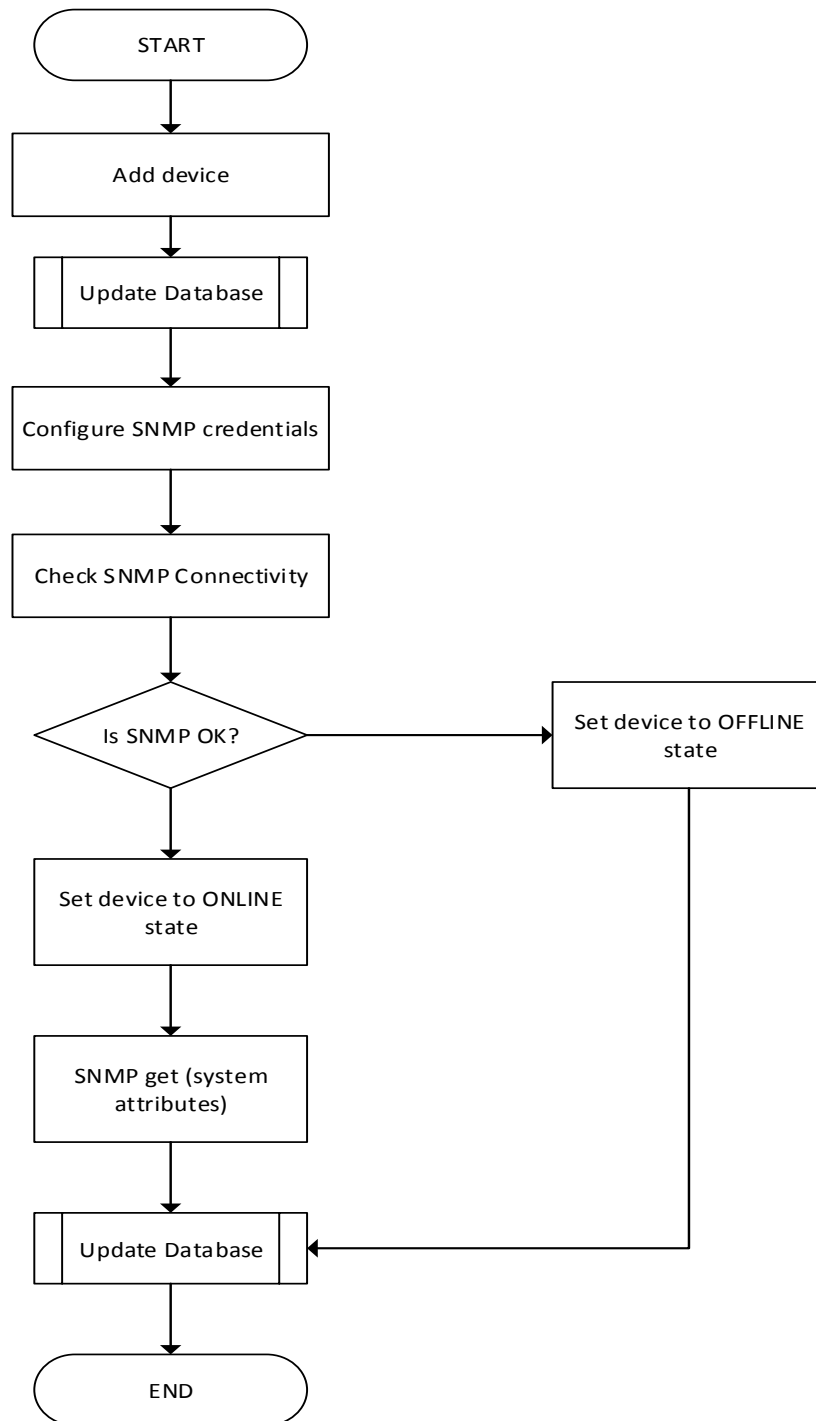


Figure 20: Device creation and information collection

4.3.1 Device Enterprise ID Collection

In order to effectively identify device manufacture, a method was created to query the enterprise OID of a device. The OID for the enterprise ID is '1.3.6.1.2.1.1.2.0'. The table below

shows the list of enterprise ID's used in the system and the logic diagram below show the flow behind the code flow. The use enterprise IDs' was the basis for achieving capability to monitor different nodes from different vendors with different sets of object identifiers (OID) and management information base (MIB) files

Table 4: Vendor Enterprise ID's

Vendor	Enterprise ID
Cisco	9
Huawei	2011
ZTE	3902

```
def get_enterprise_id(self):
    """
    Provides the enterprise ID of the device
    :return: Enterprise ID as integer
    """
    SysobjectID = self.device_session.get(oid='1.3.6.1.2.1.1.2.0') # get SysobjectID
    SysobjectID = SysobjectID.split('.')
    enterprise_id = int(SysobjectID[6])
    return enterprise_id
```

Figure 21: System Code for Enterprise ID Collection

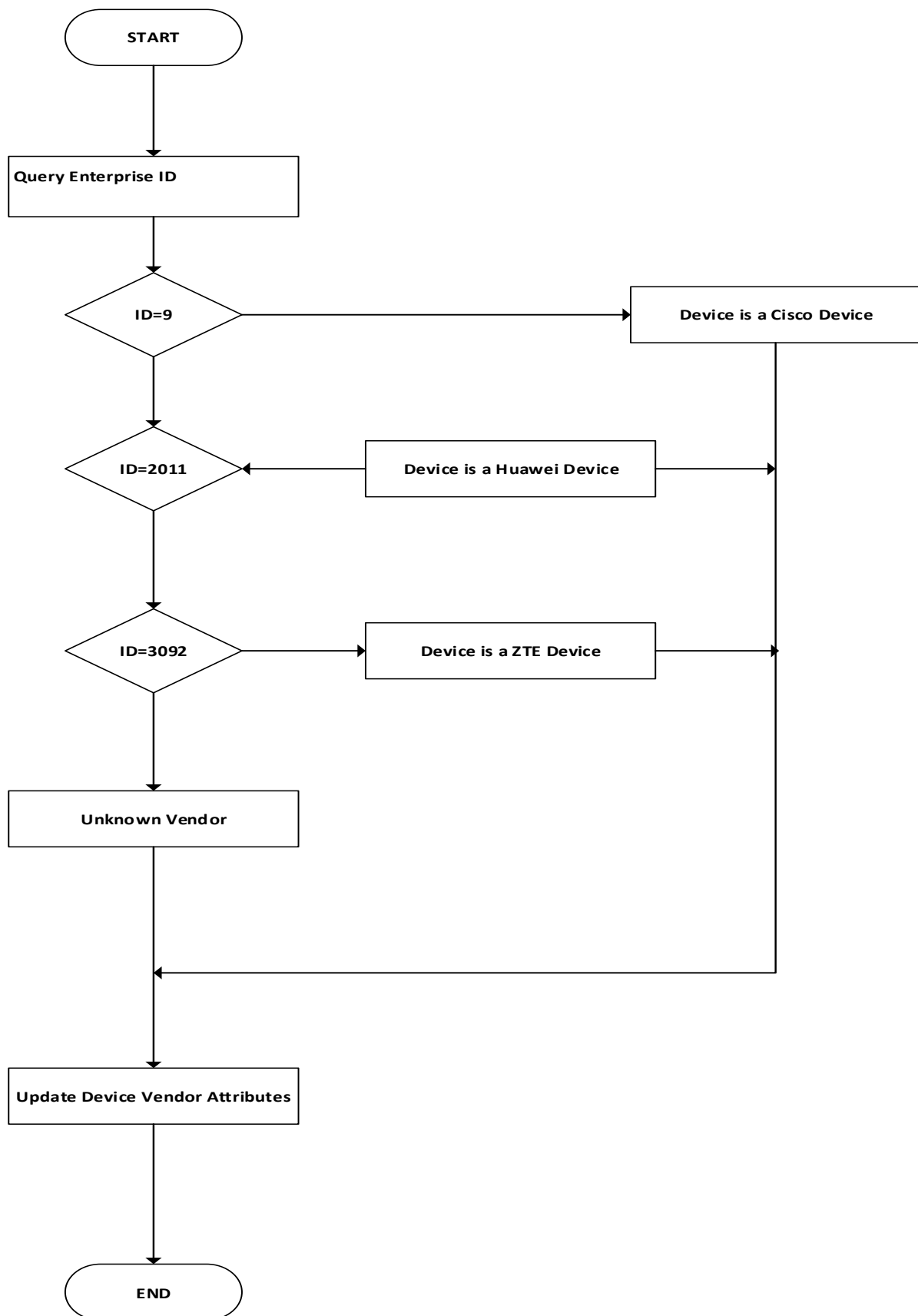


Figure 22: Design Logic for Device Enterprise ID Collection

The output result of a vendor enterprise ID detection and collection is depicted on the dashboard of the graphical user interface for network monitoring system as indicated below. For the design process, due to the cost associated with procuring different network nodes from the various vendors on the market, a maximum of three layer 3 switches were acquired and were able to be connected and monitored on the network monitoring system.

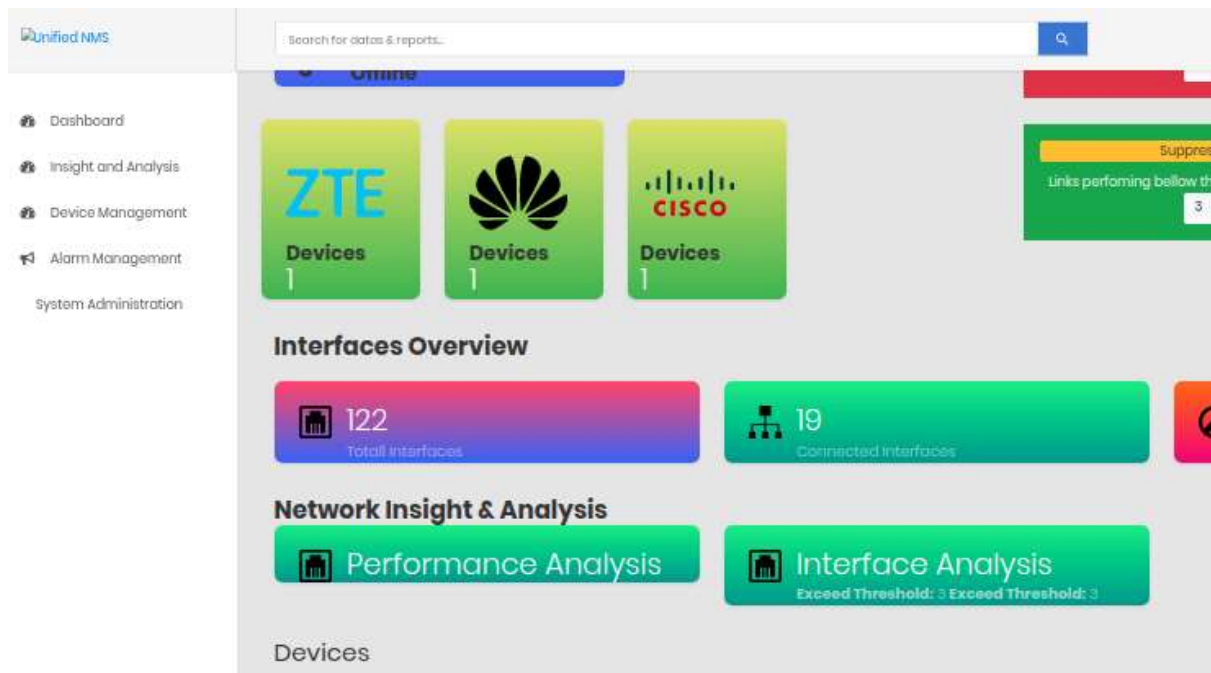


Figure 23: Vendor statistics on dashboard

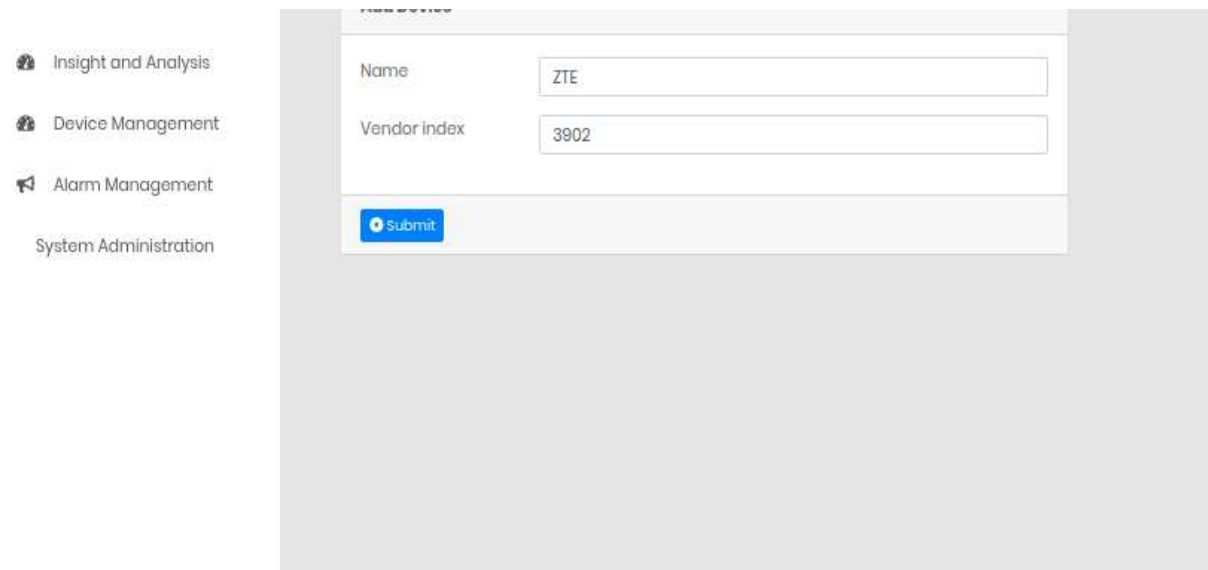
The screenshot shows the 'Vendor List' table within the system. The table lists three vendors: Cisco, Huawei, and ZTE, with their respective Vendor Index, Models, and Devices counts. Each row includes action icons for edit, delete, and refresh.

Name	Vendor Index	Models	Devices	
Cisco	8	0	1	[Edit] [Delete] [Refresh]
Huawei	2018	0	1	[Edit] [Delete] [Refresh]
ZTE	3902	0	1	[Edit] [Delete] [Refresh]

Figure 24: A detailed table for vendor details

The system was able to list vendors configured on the server and their properties. In addition the system was able to provide a count of how many devices of each vendor was being managed and of those devices managed, how many interfaces in total were being monitored.

The following image shows how to add and configure a new vendor into the system, the key element is to add the Vendor Index which will be used to identify vendors from the snmp information.



The screenshot shows a web interface for adding a new vendor. On the left is a sidebar with navigation links: 'Insight and Analysis', 'Device Management', 'Alarm Management', and 'System Administration'. The main content area is titled 'Add Vendor' and contains a form with two input fields: 'Name' with the value 'ZTE' and 'Vendor Index' with the value '3902'. Below the fields is a blue 'Submit' button.

Figure 25: Panel for creating a vendor

Separate dashboards were created for configured vendors. The images below show dashboards for ZTE, Huawei and Cisco devices.

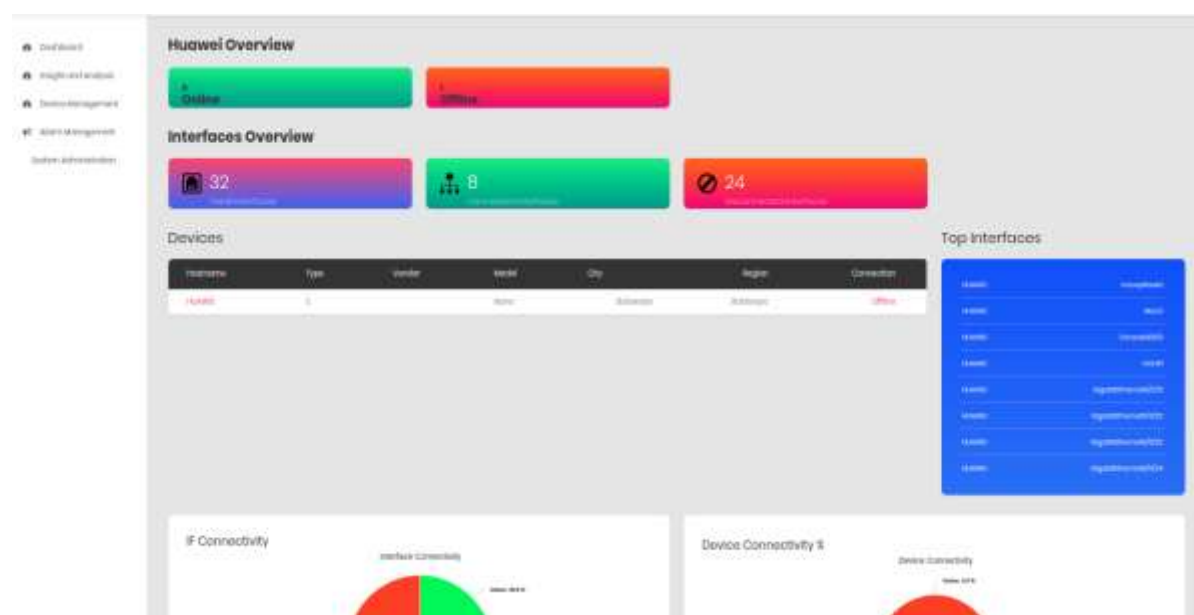


Figure 26: Huawei Network Node Overview

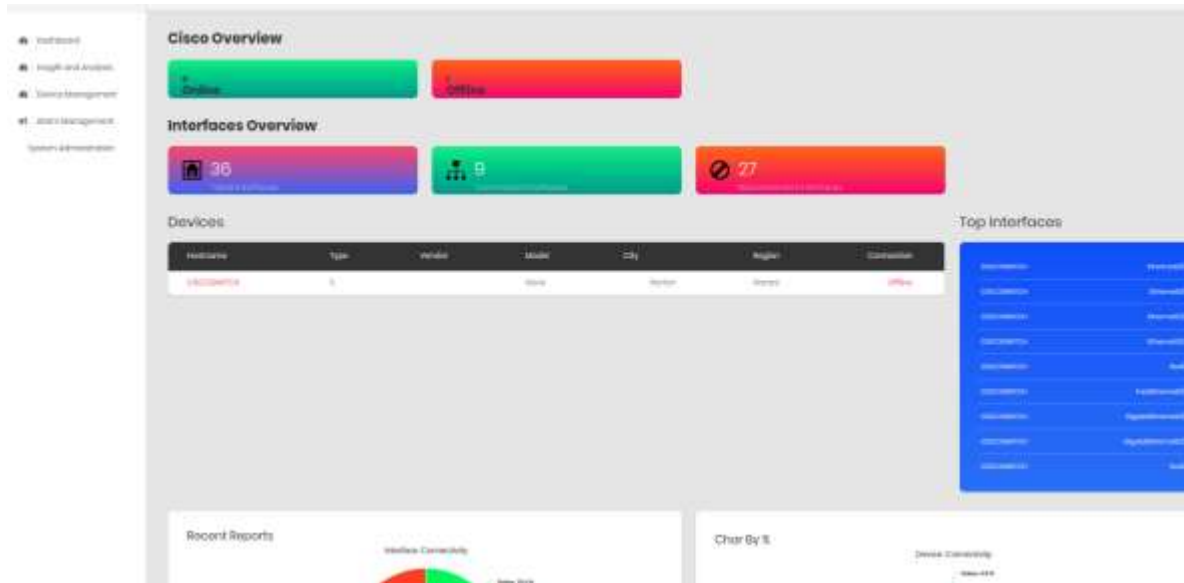


Figure 27: Cisco Network Node overview



Figure 28: ZTE Network Node Overview

4.3.2 Receiving and Parsing SNMP traps

In order to handle traps received from the NetSNMP application, I designed a module for processing the trap attributes into a format that can be used within the system for data presentation and storage. The Table 5 show the modules created for the process of handling traps.

Table 5: Modules created for the process of handling traps

Module	Role
SNMPTrapManager	A manger class for SNMP traps
SNMPTrap	An SNMP trap object which extends the SNMPTrapManager class. This object holds the attributes of a trap
InterfaceTrap	Model class for traps related to interfaces

Table 6: Methods created to save and process traps.

Module	Method	Description
SNMPTrapManager	Savetrap	Used to save a trap in to the database
	__set_trap_model	Used to check if the trap belongs to any of recognised types by the system
	__check_interface_trap	Used to check if a similar trap was received before or not
	__interfacetrap_handler	Used to process traps that belong to an interface
	__interface_status	Used to check if an interface is UP or DOWN.
SNMPTrap	retrieve_key	Retrieve the key value used to identify the type of trap received.
	Process_IP_from_UDP	In the case where the IP was not in the varbinds, collect the IP address from the UDP attributes
	__ip_key	Retrieve the IP from the received dictionary
	__oid	Retrieve the OID from the received trap dictionary
	Time	Retrieve the time in the trap received in the dictionary
	interface_index	Identify the interface index value from the received dictionary
	Device	Method to get the corresponding device object based on the host IP received

The flow chart in Figure 29 shows the code logic, flow and steps in processing traps.

From the flowchart below, when a trap is received the system checks the OID of the trap. It checks to see if the OID represents IFDown or IFUp. After checking the OID, the system identifies the index of the interface and the IP address of the device. The IP address is extracted from the source IP of the trap. The system also checks to see if the device is configured in the system and if the IP address is not defined in the system it is discarded right away. If the system successfully identifies the IP address of the device it then checks if a record of the trap exists in the system and if the trap is a new entry it is created or if it exists it updates the record.

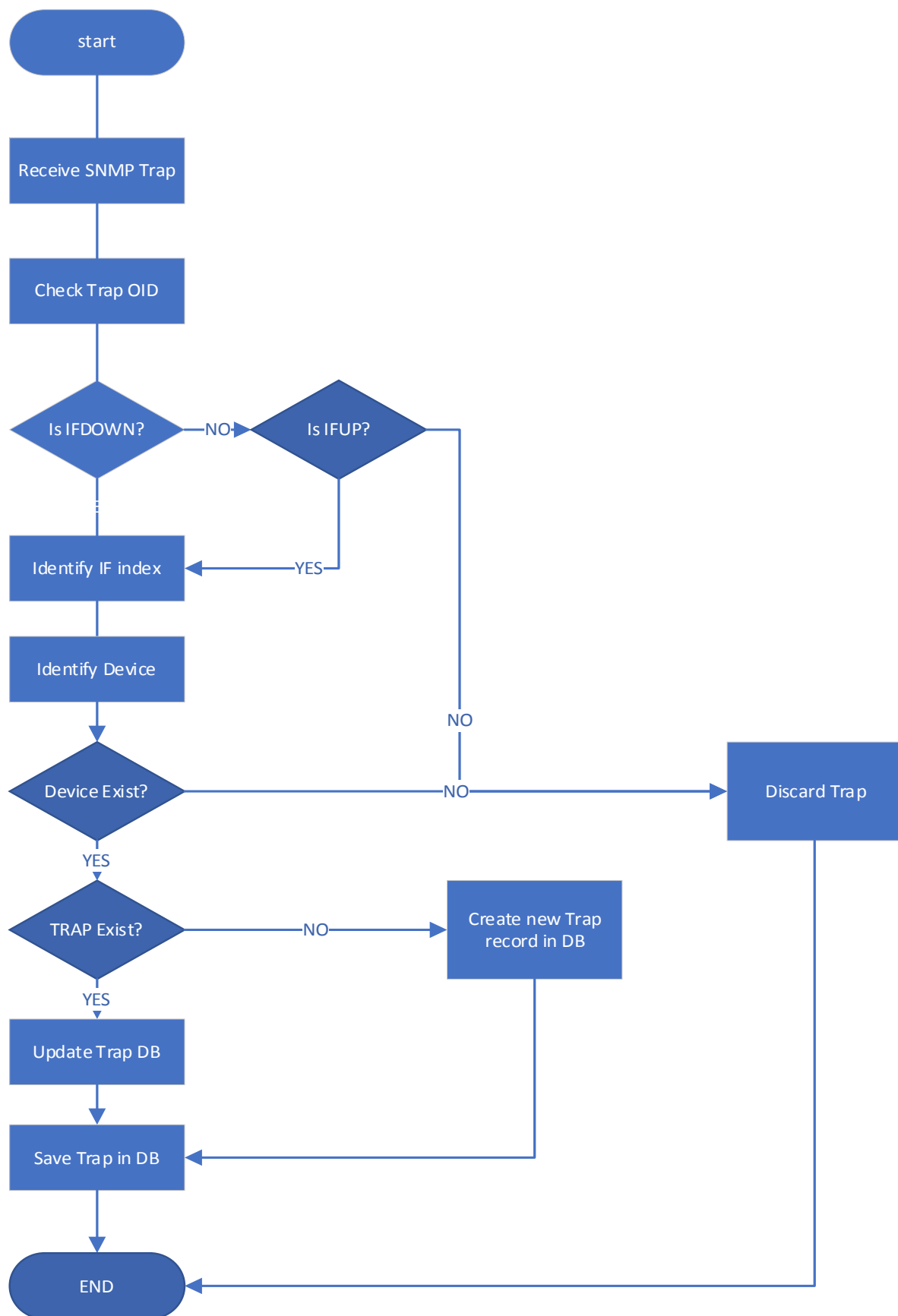


Figure 29: Trap Processing Flow diagram

The key steps in identifying a trap are below

1. Identify the trap type
2. Identify the interface index of the trap source
3. Identify the source IP of the trap, if the IP is does not belong to a device in the system the trap is considered as spam and it will be discarded.
4. Update the records in the trap database

The sample below is an example of the traps received from the SNMP application

```
{'IF-MIB::ifIndex.3': '3',  
'IF-MIB::ifType.3': 'ethernetCsmacd',  
'SNMPv2-SMI::snmpModules.18.1.3.0': '192.168.1.2',  
'SNMPv2-SMI::enterprises.9.2.2.1.1.20.3': '"up"',  
'SNMPv2-MIB::snmpTrapOID.0': 'IF-MIB::linkUp',  
'SNMPv2-SMI::snmpModules.18.1.4.0': '"public"',  
'SNMPv2-MIB::sysUpTime.0': '0:0:15:07.77',
```

The table 7 shows the keys used for trap processing.

Table 7: Tap Processing Keys

SNMP Dictionary Key	Attribute for Trap
IF-MIB::ifIndex	IF index
SNMPv2-MIB::snmpTrapOID	OID
SNMPv2-SMI::snmpModules.18.1.3.0	Source IP of the SNMP trap

Received traps are presented as alarms on the system as shown in the screenshot below, the image is showing a device interface showing an interface is disconnected.

Device	Index	Description	Status	Occur Time	Clear Time
usnmp-gw	2	Ethernet0/1	Disconnected	April 4, 2020, 4:38 p.m.	None

Figure 30: Interface disconnection alarm displayed on the User Interface

Device	Index	Description	Status	Occur Time	Clear Time
usnmp-gw	2	Ethernet0/1	Connected	April 4, 2020, 4:38 p.m.	April 4, 2020, 4:42 p.m.

Figure 31: Interface disconnection alarm cleared

4.3.3 System constraints during trap handling

During the process of handling traps, it was noted that some traps were being lost if they are received whilst the system is still processing another trap. This issue then required me to implement multi-processing technics within the system. I used Celery library to implement the multi-process feature.

The diagram below shows how the traps were handled in the two processing methods.

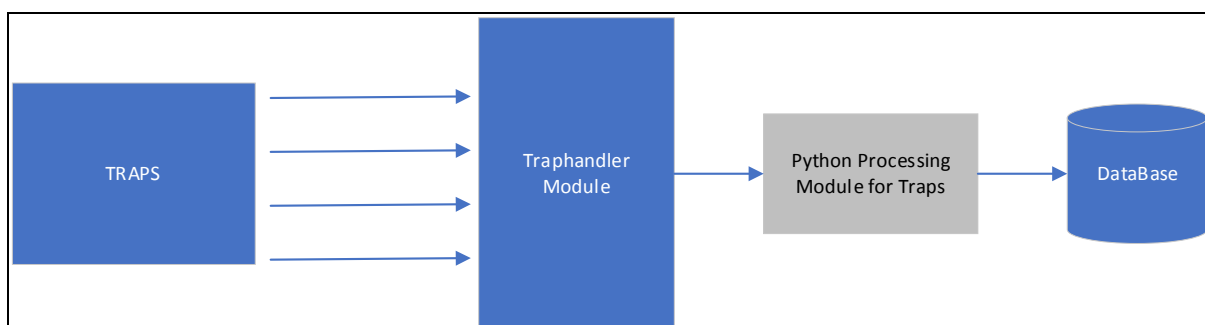


Figure 32: Synchronous trap processing

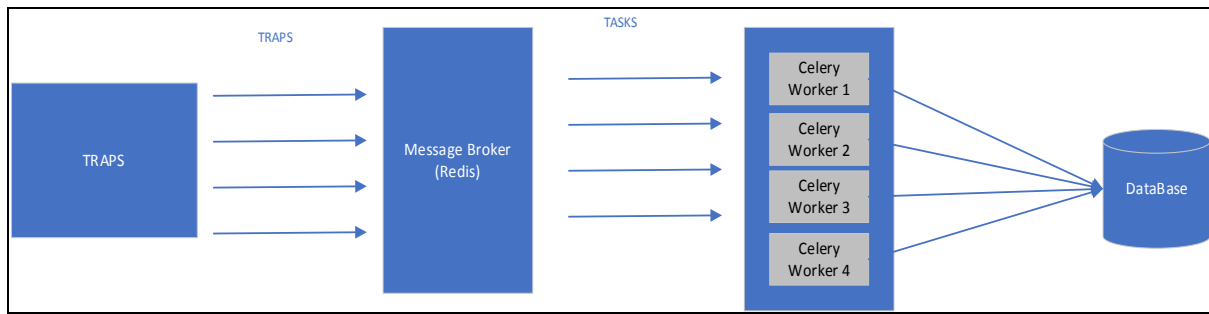


Figure 33: Asynchronous processing of traps

The introduction of Celery helped in the asynchronous processing of traps in the application.

4.3.4 Database Structure and SNMP Traps Storage

The main database used for the system design was SQLite3 mainly because of its efficiency at manipulating big datasets and also with a view of using its easier capability to migrate to some other relational database either during or after the system development. The other database used in the system development was Redis, mainly because it is an open-source, in-memory data structure store that is used as a database. Redis was used to store messages produced by the application code describing the work to be done in the Celery task queue and also served as storage of results coming off the celery queues.

The image below is showing how the traps are stored in the database.

occur_time	clear_time	interface_id	linkStatus	name
Filter	Filter	Filter	Filter	Filter
<i>NULL</i>	2020-02-20 18:28:18.458430	1	1	
<i>NULL</i>	2020-01-30 13:53:06.962261	2	1	
2020-02-10 12:25:42.301867	<i>NULL</i>	3	0	
2020-01-30 13:53:08.164348	<i>NULL</i>	4	0	
2020-01-30 13:53:08.834151	<i>NULL</i>	5	0	
2020-01-30 13:53:09.508025	<i>NULL</i>	6	0	
2020-01-30 13:53:10.014264	<i>NULL</i>	7	0	
2020-01-30 13:53:10.558873	<i>NULL</i>	8	0	
2020-02-18 21:11:32.365195	2020-02-18 21:12:01.940831	46	1	
<i>NULL</i>	2020-02-18 21:14:51.645236	47	1	
2020-02-18 21:12:51.479162	2020-02-20 18:27:12.282233	124	1	
<i>NULL</i>	2020-02-18 21:20:54.991821	125	1	

Figure 34: Traps stored in the database

4.3.5 Retrieving SNMP attributes from a database and pass them to SNMP libraries/tools

The system was supposed to be able to store SNMP details for each device. This information is to be used by the system when establishing SNMP connections with devices. Some of the SNMP attributes for the devices that were captured and processed by the system are listed in the Table 8

Table 8: Fig: List of Devices' SNMP Attributes

Attribute	Description
Version	SNMP version supported
Read Community	Read community string configured on device
Write Community	Write community string configured on device
Timeout	Snmp timeout interval
Security Level	no_auth_or_privacy, auth_with_privacy and auth_without_privacy

A class was created to hold the attributes of each device. The class contains the logic and methods used to query and modify the attributes. The UML diagram show the relationship of the device and the snmp attributes.

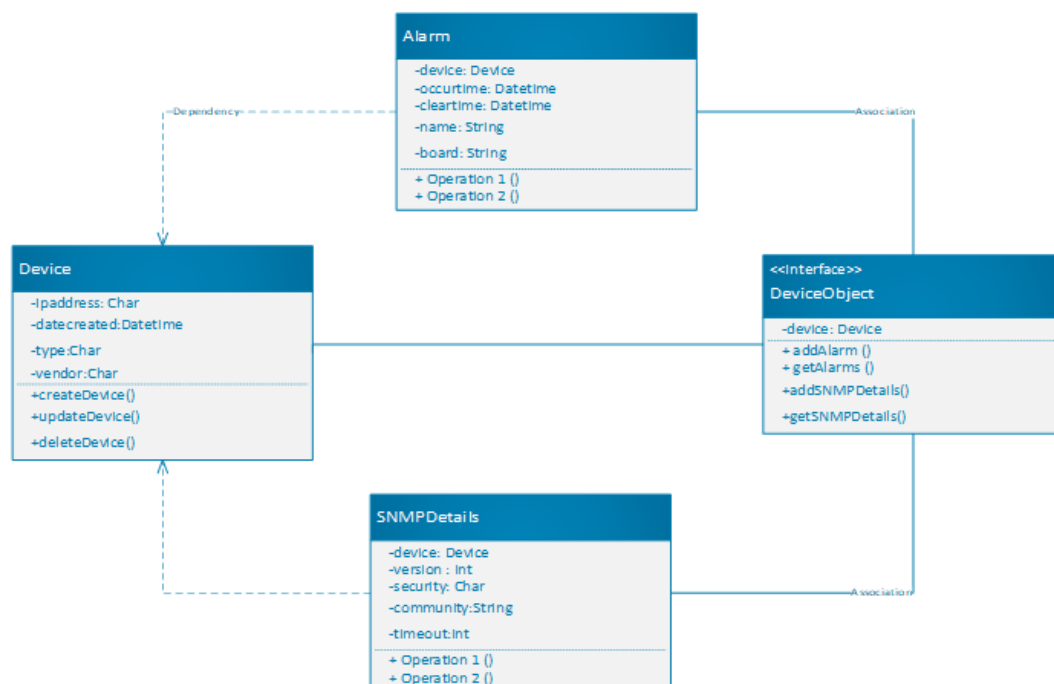


Figure 36: Relationship of device and the snmp attributes

The Figure 35 show how SNMP details are displayed in the system when retrieved from the database.

The screenshot shows the 'Unified NMS' web interface. On the left is a sidebar with navigation links: Dashboard, Insight and Analysis, Device Management, Alarm Management, and System Administration. The top of the main area has a search bar labeled 'Search for datas & reports...'. The central part of the screen displays a form titled 'CISCO SWITCH snmp details'. The form contains the following fields:

- Rcommunity: public
- Wcommunity: public
- Version: 1 (dropdown menu)
- Timeout: 5
- Retries: 5
- Security level: no_auth_or_privacy (dropdown menu)

A blue 'Submit' button is located at the bottom left of the form.

Figure 37: SNMP v2 details for a device

4.3.6 Task Scheduling

One of the requirements of the NMS system is to carry out scheduled tasks unattended. Scheduled tasks included collection of performance data, heartbeat between NE's and NMS and the updating of device information.

Table 9: Task Scheduling

Task	Schedule(s)	Description
Collect performance	15	Collect performances configured on devices
Polling	10	Check if a device is online
Update device information	30	Update any changes on device configuration

Task scheduling was achieved by configuring Celery beat schedules on the NMS. The code snippet below shows how the scheduled tasks were configured in the settings.py file of the project.

```

CELERY_BEAT_SCHEDULE = {
    'get-device-heartbeat-every-10-seconds':{
        'task': 'uNMS.tasks.check_all_device_heartbeat',
        'schedule':10.0,
    },
    'update-device-interface-every-30-seconds': {
        'task': 'uNMS.tasks.update_all_device_int',
        'schedule': 30.0,
    },
    'update_all_device_performance': {
        'task': 'uNMS.tasks.update_all_device_performance',
        'schedule': 15.0,
    }
}

```

Figure 38: Celery Beat Schedule configuration

Each task in the snapshot above has a method configured in the task module of the project. The image below shows the functions in the task module created to meet all our scheduling requirements.

```

from __future__ import unicode_literals, absolute_import
from uNMS.celery import app
from uNMS.celery import Session, exceptions
from uNMS.library.uNMS_manager import Manager
from uNMS.library import DeviceObject, SNMPtrap

@app.task(bind=True, throws=(exceptions.SMPPTimeout,), max_retries=5, default_retry_delay=5)
def list_interfaces(host):
    ...

@app.task()
def check_all_device_heartbeat():
    ...

@app.task(bind=True, throws=(exceptions.SMPPTimeout,), max_retries=5, default_retry_delay=5)
def device_heartbeat(host):
    ...

@app.task()
def save_trap(data):
    ...

@app.task()
def device_update_int(host):
    ...

@app.task()
def update_all_device_int():
    ...

@app.task()
def update_all_device_performance():
    ...

@app.task()
def collect_device_performance(host):
    ...

```

Figure 39: Task module functions

The list below describes all the methods created for scheduled tasks in the system.

- `check_all_device_heartbeat` : Used to poll all devices if they are online
- `device_heratbeat`: Used to poll a single device heartbeat
- `save trap` : Used to save a trap into the database
- `device_update_int` : used to update the interface attribute of a device
- `update_all_device_int` : used to update the interface information of all devices
- `update_all_device_performance` : used to update the device performance data for all performance instances
- `collect_device_performace` : collects the performance of each device using SNMP

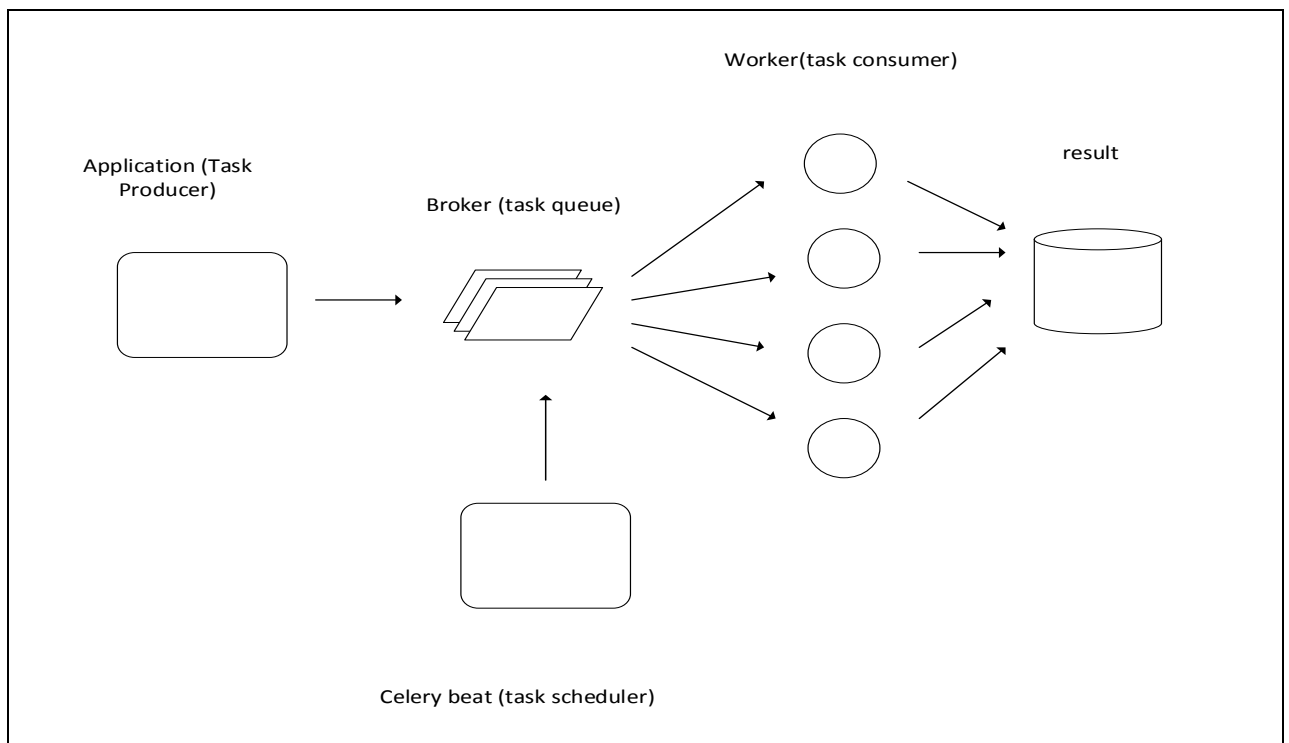


Figure 40: Celery beat schedule block diagram

4.3.7 Challenges faced in running scheduled tasks.

Running scheduled tasks presented new challenges to the design of the system. The most visible challenge was the delay in performance data collection. The system was getting slow in processing active performance data collection requests and in addition, starting any new requests to the system was also affected. Due to hardware resource limitations it was not possible to be able to create many Celery workers to handle multiple tasks at once. It was noted that some performance collection tasks took longer to finish to collect.

The ideal solution to the delay in task execution, that would maintain the quality of experience of using the system, would have been to deploy more celery workers and increase the process pool for each worker. However due to limited hardware resources we opted for reducing the frequency of performance information collection. This was a work around which enabled the achievement of a desired goal but would not be ideal if the system was to be developed for an enterprise deployment as this would negatively affect the mean time to detect (MTTD) KPI for network challenges for the service maintenance teams using the system.

Table 10: Performance information collection

Task	Schedule(s)	Description
Collect performance	60	Collect performances configured on devices
Polling	120	Check if a device is online
Update device information	300	Update any changes on device configuration

Justification of adjusting the schedules

Collection of performance data was changed to 1 minute because data throughput does not go through very frequent changes. The interval is okay for general data trending. More frequent data collection is best collected on request to minimize system resource utilisations. The Polling of device heartbeat was changed to 120 seconds, the interval is sufficient to establish if a device is online. The updating of device information was changed to 300 seconds on the assumption that configuration changes do not occur very frequent. In most industrial solutions configurations are manually synchronised.

4.3.8 Device Heartbeat Polling

One of the objectives of the system is to be able to send periodic heartbeats to the devices and ensure that communication exists between the NMS and the devices. This action is configured as scheduled task using Celery beat scheduler. The outcome of this process is to establish if a device is online or offline and update the status in the database.

The code snippet in Figure 42 shows how the polling functions were written.

```

@app.task(autoretry_for=(exceptions.SNMPTIMEOUT,),max_retries=5, default_retry_delay=6)
def device_heartbeat(host):

    device = Manager(host)
    try :
        sysname = device.systemname()

    except exceptions.SNMPInvalidAddress:
        return {host:'INVALID'}

    except exceptions.SNMPError:
        device.device_obj.setOffline()
        return {host:'ERROR'}

    except exceptions.SNMPTIMEOUT:
        device.device_obj.setOffline()
        return {host: 'OFFLINE'}

    device.device_obj.setOnline()
    return {host:'ONLINE'}

```

Figure 41: Polling Functions Code Snippet

The flow chart in Figure 43 shows the heartbeat polling process

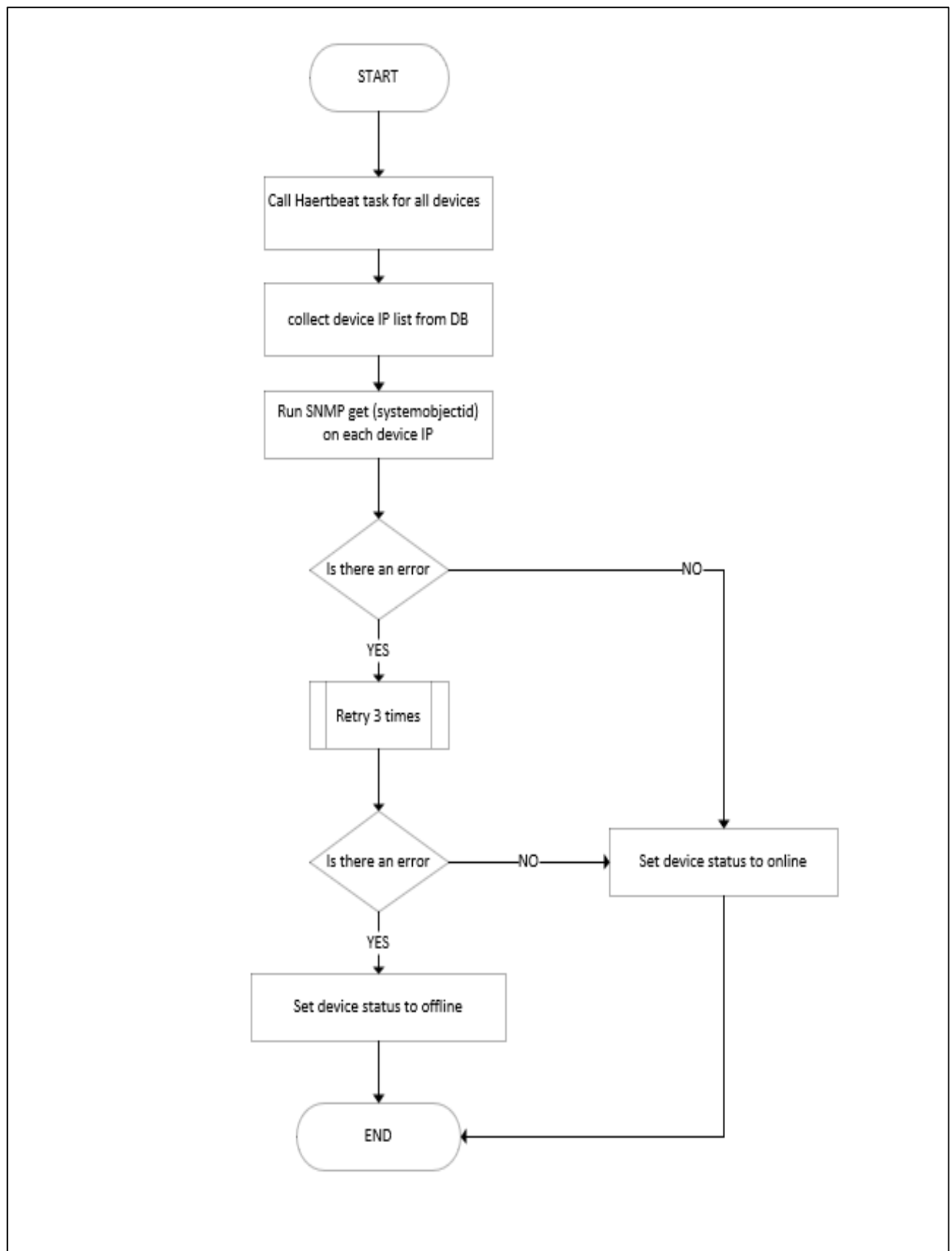
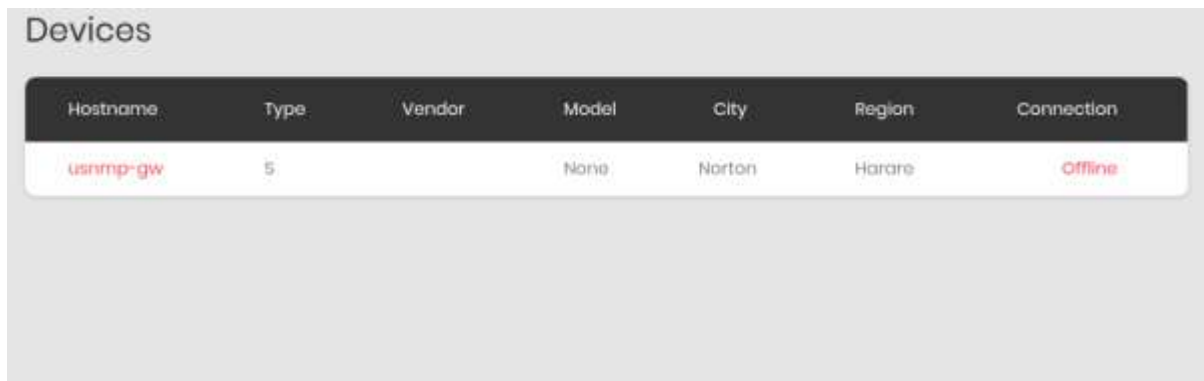


Figure 42: The process of polling the heartbeat of a device

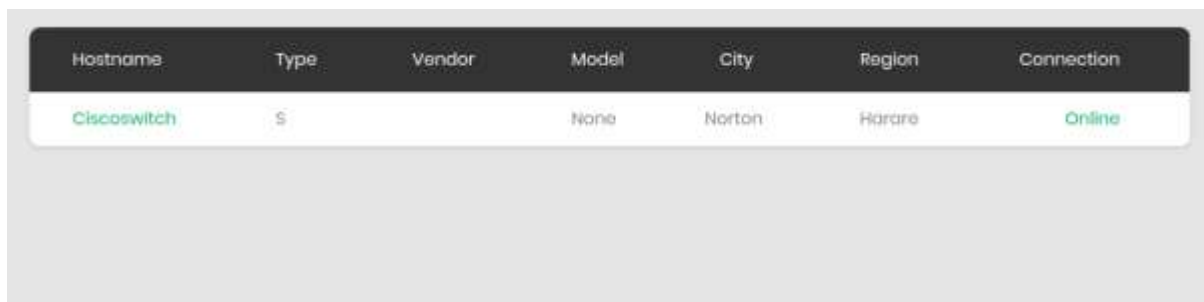
After the polling is complete the device information is update as show in the images below from the NMS user interface.



The screenshot shows a table titled "Devices" with the following columns: Hostname, Type, Vendor, Model, City, Region, and Connection. A single row is displayed with the following data: Hostname: usnmp-gw, Type: S, Vendor: None, Model: None, City: Norton, Region: Harare, and Connection: Offline.

Hostname	Type	Vendor	Model	City	Region	Connection
usnmp-gw	S		None	Norton	Harare	Offline

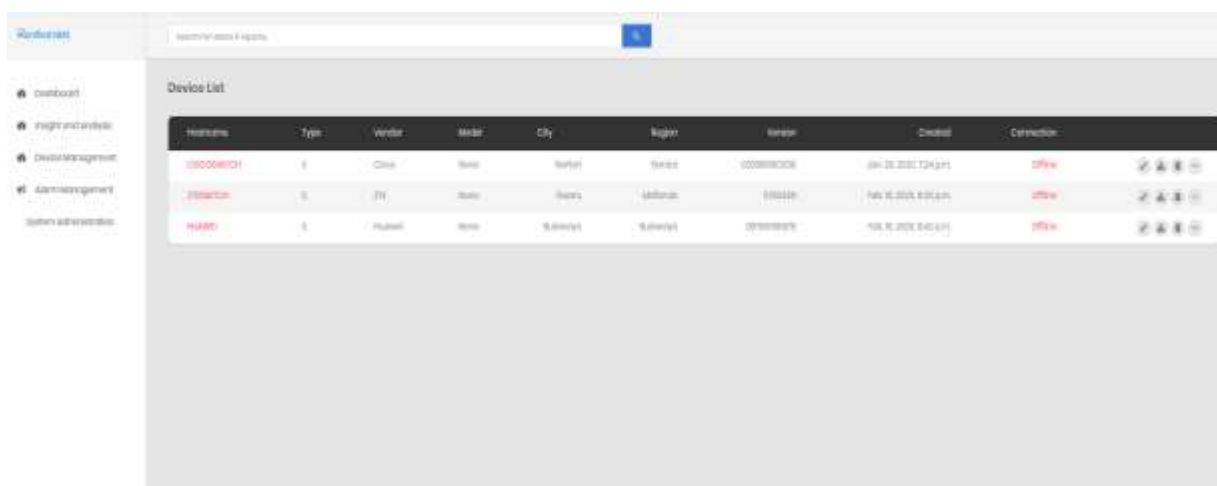
Figure 43: Device with failed heartbeat



The screenshot shows a table titled "Devices" with the following columns: Hostname, Type, Vendor, Model, City, Region, and Connection. A single row is displayed with the following data: Hostname: Ciscoswitch, Type: S, Vendor: None, Model: None, City: Norton, Region: Harare, and Connection: Online.

Hostname	Type	Vendor	Model	City	Region	Connection
Ciscoswitch	S		None	Norton	Harare	Online

Figure 44: Device with a successful heartbeat



The screenshot shows a "Device List" table with the following columns: Hostname, Type, Vendor, Model, City, Region, Version, Created, and Connection. Three rows are displayed with the following data:

Hostname	Type	Vendor	Model	City	Region	Version	Created	Connection
CISCO00000001	S	Cisco	None	Norton	Harare	CISCO00000001	Jan 26, 2023, 12:43:01	Offline
2500000000	S	HP	None	Harare	Harare	2500000000	Feb 10, 2023, 8:25:01	Offline
HUB001	S	Huawei	None	Harare	Harare	2500000000	Feb 10, 2023, 8:40:01	Offline

Figure 45: Device Listing

4.3.9 Interface Status Polling

The collection of interface data involves sending SNMP requests to all configured devices and updating the database. This action is done at specific intervals hence it was designed as a scheduled task using Celery heartbeat schedule. The flow chart below shows the process of querying interface information.

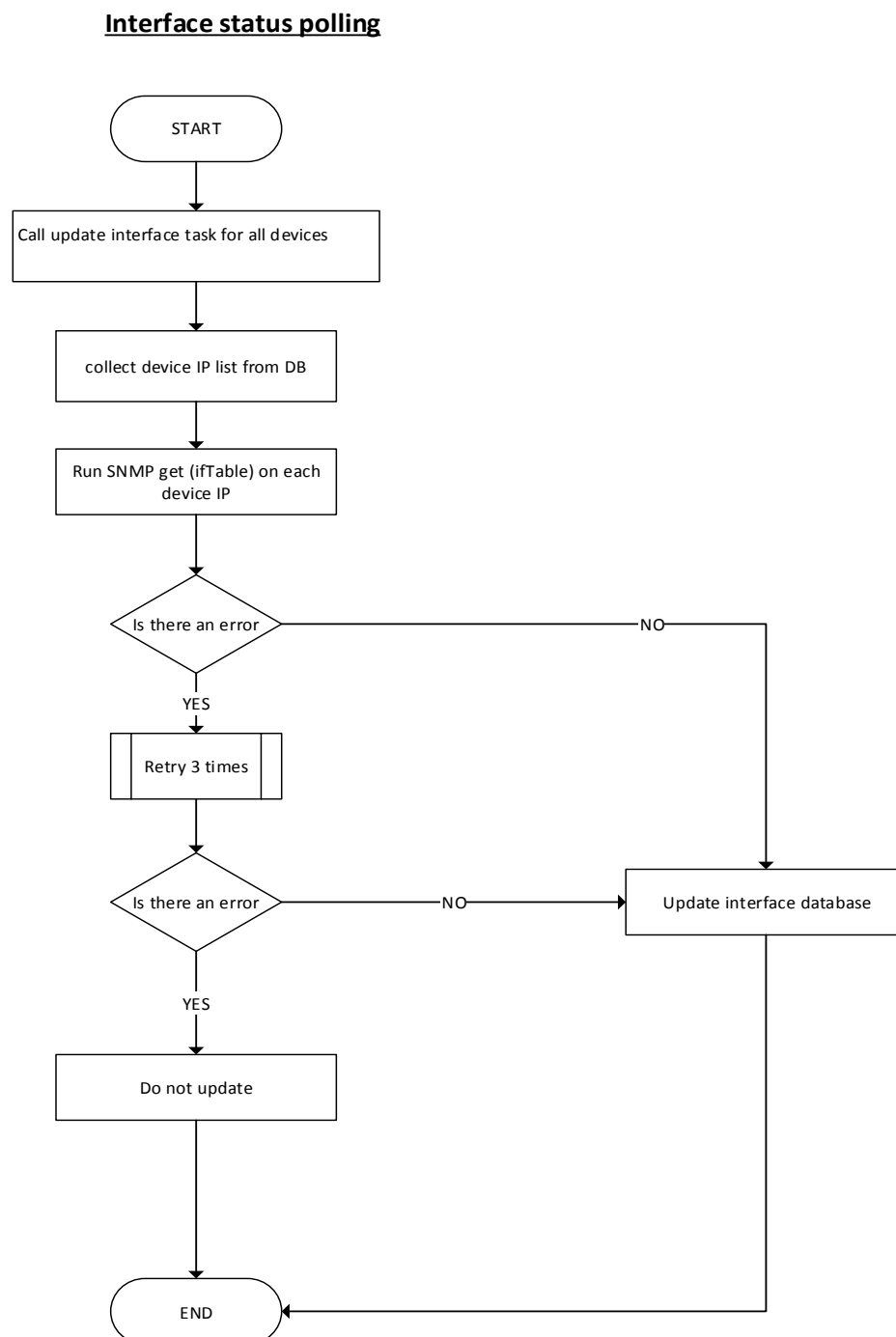


Figure 46: Interface Polling Status Flow Chart

The code snippet below shows the scheduled tasks for interface information polling.

```
@app.task()
def device_update_int(host):
    """
    Task to update device properties in the database
    """
    device = Manager(host)
    device.update_interface_db() # update the interface DB

@app.task()
def update_all_device_int():
    device_manager = DeviceObject() # create a device object manager instance
    devices = device_manager.all() # query all devices configured in the system

    for device in devices:
        device_update_int.delay(device.ipaddress) # call interface update task per device
```

Figure 47: Interface information polling code snippet

Information returned from interface polling is shown in the table below.

	id	device_ip	ifType	ifIndex	ifDescription	ifPhysAddress	ifAdminStatus	ifOperStatus	ifSpeed	ifMtu	device_id
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
63	99	192.168.1.1	192.168.1.1	56	vlan1	0:19:c0:0:fe:39	True	1	1000000000	1500	5
64	100	192.168.1.1	192.168.1.1	1	inLoopBack0	0:0:0:0:0:0	True	1	0	1500	6
65	101	192.168.1.1	192.168.1.1	2	NULL0	0:0:0:0:0:0	True	1	0	1500	6
66	102	192.168.1.1	192.168.1.1	3	Console9/0/0	0:0:0:0:0:0	True	1	0	0	6
67	103	192.168.1.1	192.168.1.1	4	Vlanif1	c0:bf:c0:a1:56:1c	True	1	1000000000	1500	6
68	104	192.168.1.1	192.168.1.1	5	GigabitEthernet0/0/1	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
69	105	192.168.1.1	192.168.1.1	6	GigabitEthernet0/0/2	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
70	106	192.168.1.1	192.168.1.1	7	GigabitEthernet0/0/3	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
71	107	192.168.1.1	192.168.1.1	8	GigabitEthernet0/0/4	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
72	108	192.168.1.1	192.168.1.1	9	GigabitEthernet0/0/5	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
73	109	192.168.1.1	192.168.1.1	10	GigabitEthernet0/0/6	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
74	110	192.168.1.1	192.168.1.1	11	GigabitEthernet0/0/7	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
75	111	192.168.1.1	192.168.1.1	12	GigabitEthernet0/0/8	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
76	112	192.168.1.1	192.168.1.1	13	GigabitEthernet0/0/9	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
77	113	192.168.1.1	192.168.1.1	14	GigabitEthernet0/0/10	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
78	114	192.168.1.1	192.168.1.1	15	GigabitEthernet0/0/11	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
79	115	192.168.1.1	192.168.1.1	16	GigabitEthernet0/0/12	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
80	116	192.168.1.1	192.168.1.1	17	GigabitEthernet0/0/13	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
81	117	192.168.1.1	192.168.1.1	18	GigabitEthernet0/0/14	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
82	118	192.168.1.1	192.168.1.1	19	GigabitEthernet0/0/15	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
83	119	192.168.1.1	192.168.1.1	20	GigabitEthernet0/0/16	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
84	120	192.168.1.1	192.168.1.1	21	GigabitEthernet0/0/17	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
85	121	192.168.1.1	192.168.1.1	22	GigabitEthernet0/0/18	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
86	122	192.168.1.1	192.168.1.1	23	GigabitEthernet0/0/19	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
87	123	192.168.1.1	192.168.1.1	24	GigabitEthernet0/0/20	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
88	124	192.168.1.1	192.168.1.1	25	GigabitEthernet0/0/21	c0:bf:c0:a1:56:1c	True	1	1000000000	1500	6
89	125	192.168.1.1	192.168.1.1	26	GigabitEthernet0/0/22	c0:bf:c0:a1:56:1c	True	1	1000000000	1500	6
90	126	192.168.1.1	192.168.1.1	27	GigabitEthernet0/0/23	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
91	127	192.168.1.1	192.168.1.1	28	GigabitEthernet0/0/24	c0:bf:c0:a1:56:1c	True	0	1000000000	1500	6
92	128	192.168.1.1	192.168.1.1	29	XGigabitEthernet0/0/1	c0:bf:c0:a1:56:1c	True	0	4294967295	1500	6
93	129	192.168.1.1	192.168.1.1	30	XGigabitEthernet0/0/2	c0:bf:c0:a1:56:1c	True	0	4294967295	1500	6
94	130	192.168.1.1	192.168.1.1	31	XGigabitEthernet0/0/3	c0:bf:c0:a1:56:1c	True	0	4294967295	1500	6

Figure 48: Interface database table

The following image is the result of the processing of the interface information collected from devices.

<div>Dashboard</div> <div>Report and Analysis</div> <div>Device Management</div> <div>Alert Management</div> <div>System Administration</div>	Interfaces List									
	ID	Device	Type	Description	MAC Address	Speed	MTU	Admin Status	Operational Status	Action
	1	SW001	Internal/Console	Ethernet0/1	00:1B:3C:3D:3E:3F	1000000	9000	Up	Up	Monitor
	2	SW002	Internal/Console	Ethernet0/2	00:1B:3C:3D:3E:40	1000000	9000	Up	Up	Monitor
	3	SW003	Internal/Console	Ethernet0/3	00:1B:3C:3D:3E:41	1000000	9000	Up	Up	Monitor
System Administration	4	SW004	Internal/Console	Ethernet0/4	00:1B:3C:3D:3E:42	1000000	9000	Up	Up	Monitor
	5	SW005	Internal/Console	Ethernet0/5	00:1B:3C:3D:3E:43	1000000	9000	Up	Down	Monitor
	6	SW006	Internal/Console	Ethernet0/6	00:1B:3C:3D:3E:44	1000000	9000	Up	Down	Monitor
	7	SW007	Internal/Console	Ethernet0/7	00:1B:3C:3D:3E:45	1000000	9000	Up	Down	Monitor
	8	SW008	Internal/Console	Ethernet0/8	00:1B:3C:3D:3E:46	1000000	9000	Up	Down	Monitor
	9	SW009	Other	Null	00:00:00:00:00:00	100000000	9000	Up	Up	Monitor
	1000	SW010	None	FastEthernet0/24	00:1B:3C:3D:3E:47	100000000	9000	Up	Up	Monitor
	1001	SW011	None	FastEthernet0/25	00:1B:3C:3D:3E:48	100000000	9000	Up	Down	Monitor
	1002	SW012	None	FastEthernet0/26	00:1B:3C:3D:3E:49	100000000	9000	Up	Down	Monitor
	1003	SW013	None	FastEthernet0/27	00:1B:3C:3D:3E:4A	100000000	9000	Up	Down	Monitor

Figure 49: Interface listing on the NMS

4.3.10 Interface Performance Collection

Interface performance collection is one of the objectives of the system design. In the implementation of this requirement, a flow of events was established first in order to come up with a functional collection of information. The flow chart below shows the sequence of events for collecting performance data.

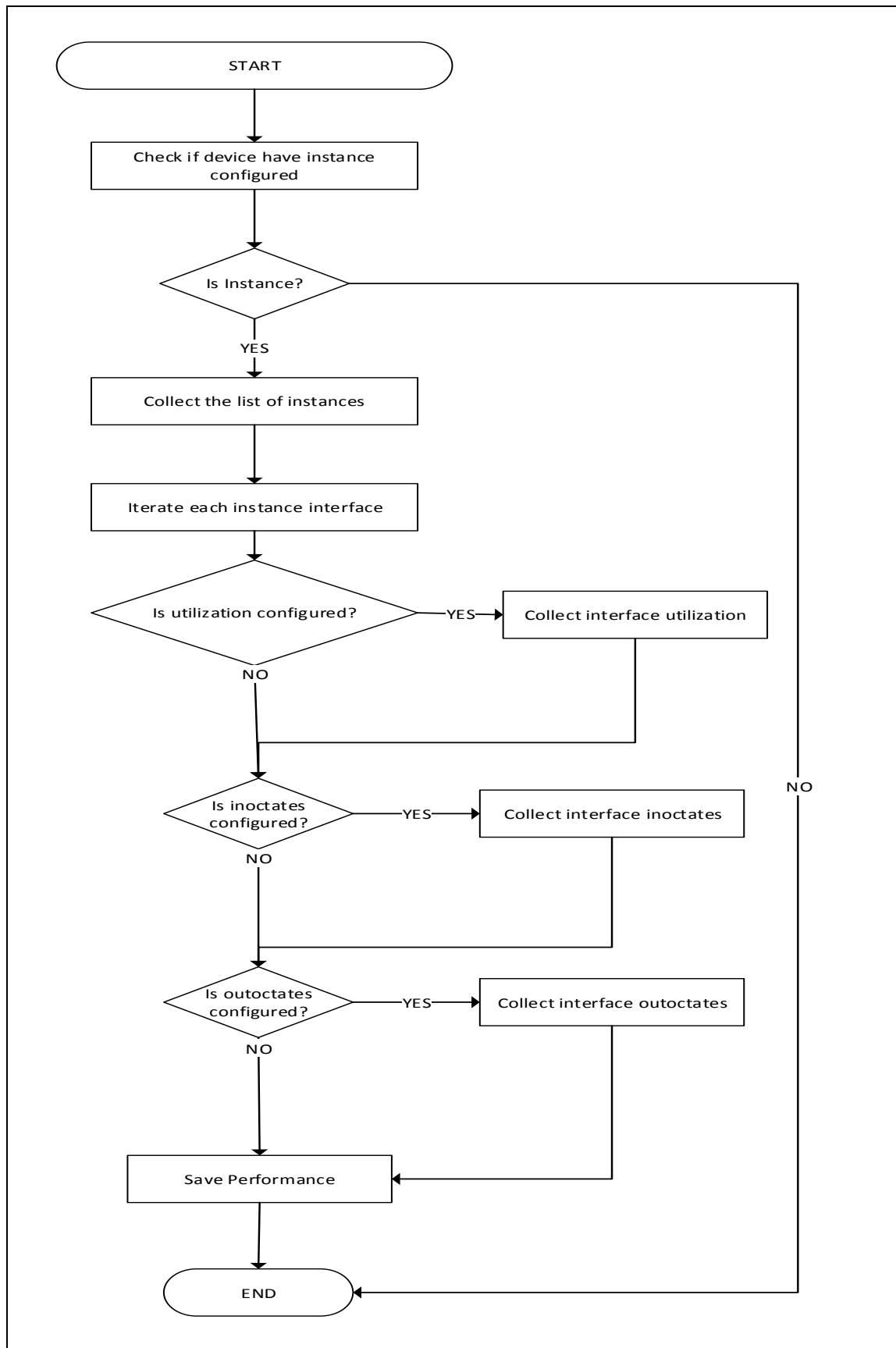


Figure 50: Performance data collection flow chart

4.3.11 Collecting and Storing Performance Data.

In order to process performance data a class was created to handle parse and save performance data collected using SNMP tools. The table below list the methods and purposes for each method in collecting and processing performance data.

Table 11: Processing performance data

Method	Purpose
Interface_instance()	Collect interface performance instances configured for a given interface
check_if_instance_exist()	Checks if a performance instance is configured for a given interface
collect_performances()	Collects interface performance based on the instances configured and save them to the database
collectInterfaceUtilization()	Collects interface utilization performance information for a given interface
save_interface_utilization()	Save interface utilization performance into the database
save_interface_inoctates()	Save the interface inoctates performance into the database
save_interface_outoctates()	Save the interface outoctates performance into the database

4.3.11 Processing performance data

The block diagram below shows the abstract overview of the processing of the performance data. The performance manager module is responsible for initiating the collection of data and also the processing of the returned data. When data is received it is raw and it needs to be manipulated into a format that can be stored in the database.

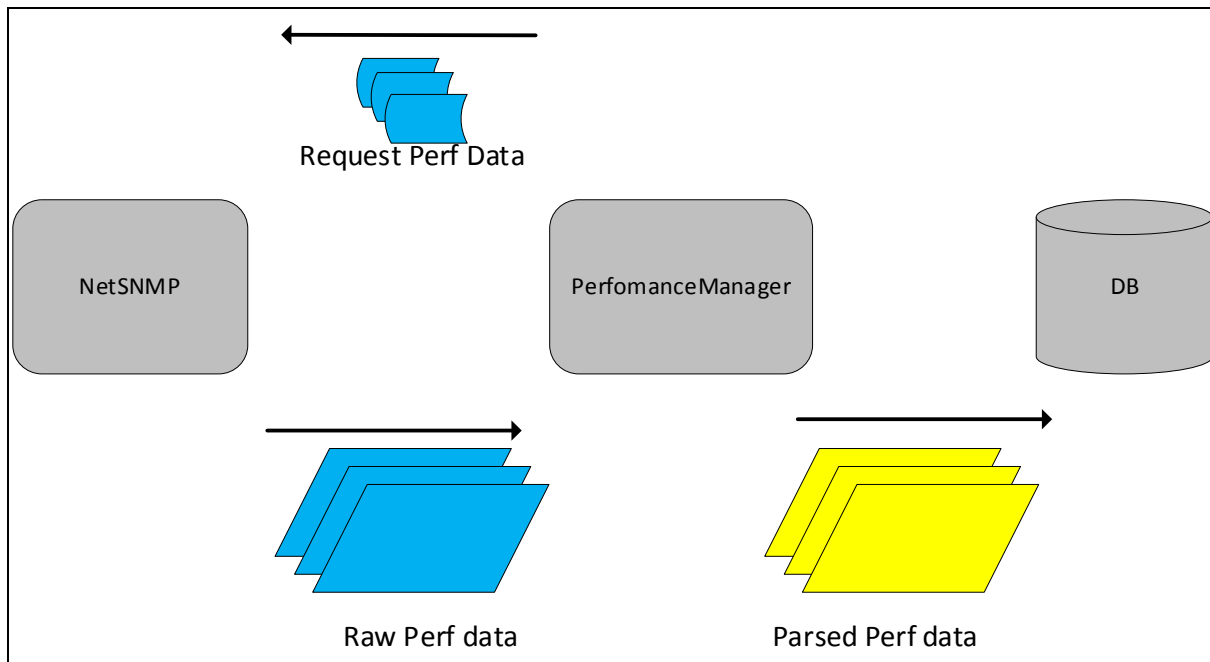


Figure 51: Performance data processing overview

The following screenshot shows the code snippet of how the PerformanceManager class is built.

```

class PerformanceManager(object):
    """
    Class to manage performance related stuff
    """

    def __init__(self, manager):...

    @property
    def interfaces_instances(self):...

    def interface_instance(self, ifIndex):...

    def check_if_instance_exist(self):...

    def collect_performances(self):...

    def collectInterfaceUtilization(self):...

    def save_interface_utilization(self, value, interface):
        InterfaceUtilization.objects.create(value=value, interface=interface, time=datetime.datetime.now())

    def save_interface_inoctates(self, value, interface):
        InterfaceInboundOctates.objects.create(value=value, interface=interface, time=datetime.datetime.now())

    def save_interface_utilization(self, value, interface):
        InterfaceOutboundOctates.objects.create(value=value, interface=interface, time=datetime.datetime.now())
  
```

Figure 52: Code snippet of how the Performance Manager class is built.

4.3.12 Presenting Performance Data

For the presentation of performance data another class called Performance Display was created. The role of the class is to handle performance information requests from the user interface and present it to the user interface. The following block diagram shows the functional building blocks for achieving the task of presenting performance data to the UI.

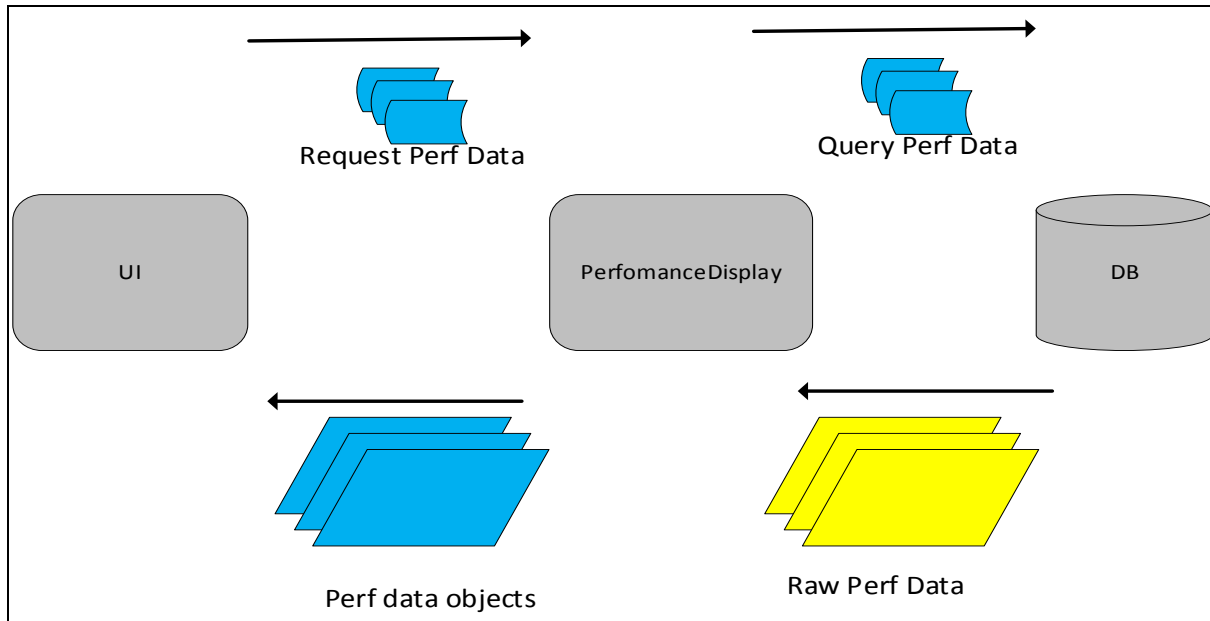


Figure 53: Building blocks for of presenting performance data to the UI

Table 12: Performance Display Attributes

Method	Purpose
parsePerformance()	Process retrieved performance data to a format that can be processed by the Django templating system
interface_utilizations	Query the database for interface utilization
interface_inoctates	Query the database for interface inoctates
interface_outoctates	Query the database for interface outoctates
interface_performances	Query all performances for all interfaces on a device from the database
interface_performance_all	Query the performances for a given interface from the database

The method `parsePerformance` is used to process data retrieved from the database to a format that is presentable to the Django templating system. The performance data need to be transformed into a two separate lists containing values and timestamps. An example of the data is shown below shows the utilization values collected for an interface `GigabitEthernet0/0/21`.

Table 13: Values collected for an interface `GigabitEthernet0/0/21`

id	value	time	interface_id
1152	4.62623535129881	2020-02-18 19:18:16.035603	GigabitEthernet0/0/21
1154	6.74307304785894	2020-02-18 19:18:31.790802	GigabitEthernet0/0/21
1156	10.0224634968177	2020-02-18 19:18:45.991290	GigabitEthernet0/0/21
1158	6.39350379890436	2020-02-18 19:19:01.770487	GigabitEthernet0/0/21
1160	25.9240386324584	2020-02-18 19:19:15.949701	GigabitEthernet0/0/21
1162	1.7119036294819	2020-02-18 19:19:31.930410	GigabitEthernet0/0/21
1163	13.8194489255985	2020-02-18 19:19:45.928581	GigabitEthernet0/0/21
1166	1.4631863383106	2020-02-18 19:20:02.004455	GigabitEthernet0/0/21
1168	25.6832763686514	2020-02-18 19:20:15.874930	GigabitEthernet0/0/21
1170	1.98076580587711	2020-02-18 19:20:31.910327	GigabitEthernet0/0/21
1172	27.1949206349206	2020-02-18 19:20:45.873365	GigabitEthernet0/0/21
1174	8.81988345036345	2020-02-18 19:21:02.080892	GigabitEthernet0/0/21
1176	24.6799193316047	2020-02-18 19:21:15.872800	GigabitEthernet0/0/21
1178	0.0357352768308452	2020-02-18 19:21:31.961101	GigabitEthernet0/0/21

The data from the table above is processed into two lists shown below.

Time =

[18:16.0, 18:31.8, 18:46.0, 19:01.8, 19:15.9, 19:31.9, 19:45.9, 20:02.0, 20:15.9, 20:31.9, 20:45.9, 21:02.1, 21:15.9, 21:32.0]

```
values = [4.626235351, 6.743073048, 10.0224635, 6.393503799,
25.92403863,1.711903629,13.81944893,1.463186338,25.68327637,1.
980765806,27.19492063,8.81988345,24.67991933,0.035735277]
```

Configuring a performance instance

The image, Figure 55 showing the interface for configuring the performance collection instance for a device interface. It shows the configuration of upper and lower thresholds, selection of indicators and the interface whose performance utilization would be desired to be measured.

Device: CISCO SWITCH
Interface: Ethernet0/1
Utilization: ☒
Inoctates: ☒
Outoctates: ☒
Lower threshold utilization: 23
Upper threshold utilization: 70
Lower threshold octoctates: 20
Upper threshold octoctates: 80
Lower threshold inoctates: 10
Upper threshold inoctates: 80
Submit

Figure 54: Configuring the performance instance

The next image, Figure 56 is showing the presentation of the performance graphs.

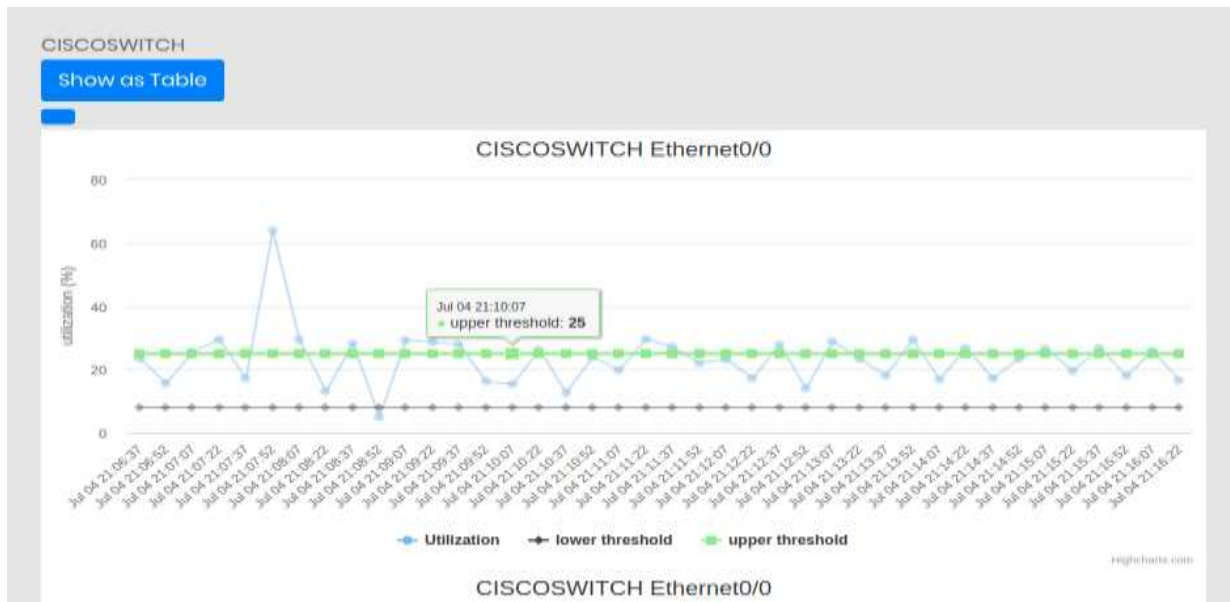


Figure 55: Interface Utilization plotted with thresholds



Figure 56: Performance graph for an interface

4.3.13 Network insight and analytics

Another objective of the NMS is to provide insight into possible faults on the network and help in making planning decisions. This part of the project mainly focused on monitoring interfaces on devices. The key performance indicators that this section looked at were the frequency an interface is in disconnected mode to address issues of link stability, high interface utilization

to address issues of interface or link congestion and very low interface utilization to address issue of degraded service or compromised link quality.

1. Link stability
2. High interface utilization
3. Low interface utilization

4.3.14 Effects of unstable interfaces

An interface that frequently disconnects affects the quality of service experienced on the link. In some cases the situation can go undetected depending on the monitoring system in place, leading to unwanted customer experience. Having a knowledge of interfaces with poor quality, assists both network operations centre (NOC) and service operation centre (SOC) personnel in achieving one of their key performance indicators which is reducing the Mean Time to Repair (MTTR) as it improves response time.

How to detect unstable interfaces

Unstable interfaces were identified based on the frequency an interface disconnects and connects. Each interface status is tracked in NMS, a separate database table was created for recording changes in interface UP statuses and the times in which they occur. For demonstration purposes a maximum of 5 disconnections per minute was used as benchmark for classifying an interface as unstable. The amount of disconnections per minute was calculated as an average of disconnections over a given period. The formula below was used to determine the disconnections per minute.

$$\text{Disconnections per minute} = \frac{\text{total disconnections}}{\text{time in minutes}}$$

Table 14: Interface stability calculations in an interval of 10 minutes

Interface	Disconnection Count	T of first disconnection	T of last disconnection	Disconnections per minute
GigabitEthernet0/0/21	28	2020-02-18 19:19	2020-02-18 19:29	2.8
GigabitEthernet0/0/22	0	2020-02-18 19:19	2020-02-18 19:29	0.0
GigabitEthernet0/0/23	1	2020-02-18 19:19	2020-02-18 19:29	0.1
GigabitEthernet0/0/24	3	2020-02-18 19:19	2020-02-18 19:29	0.3
GigabitEthernet0/0/25	8	2020-02-18 19:19	2020-02-18 19:29	0.8
GigabitEthernet0/0/26	9	2020-02-18 19:19	2020-02-18 19:29	0.9
GigabitEthernet0/0/27	10	2020-02-18 19:19	2020-02-18 19:29	1.0
GigabitEthernet0/0/28	23	2020-02-18 19:19	2020-02-18 19:29	2.3

The network insight were summarised and presented as part of the dashboard. The image below shows interface analytics summary. Links of high utilization are shown as congested.

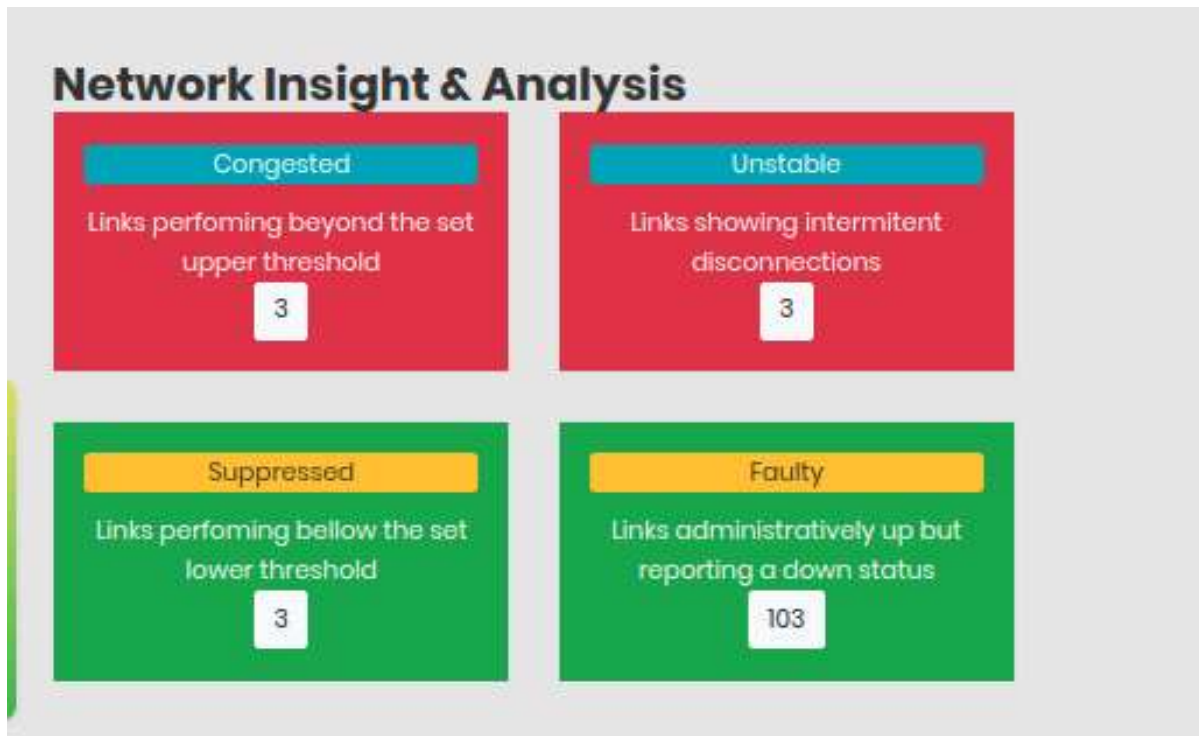


Figure 57: Network Insight dashboard

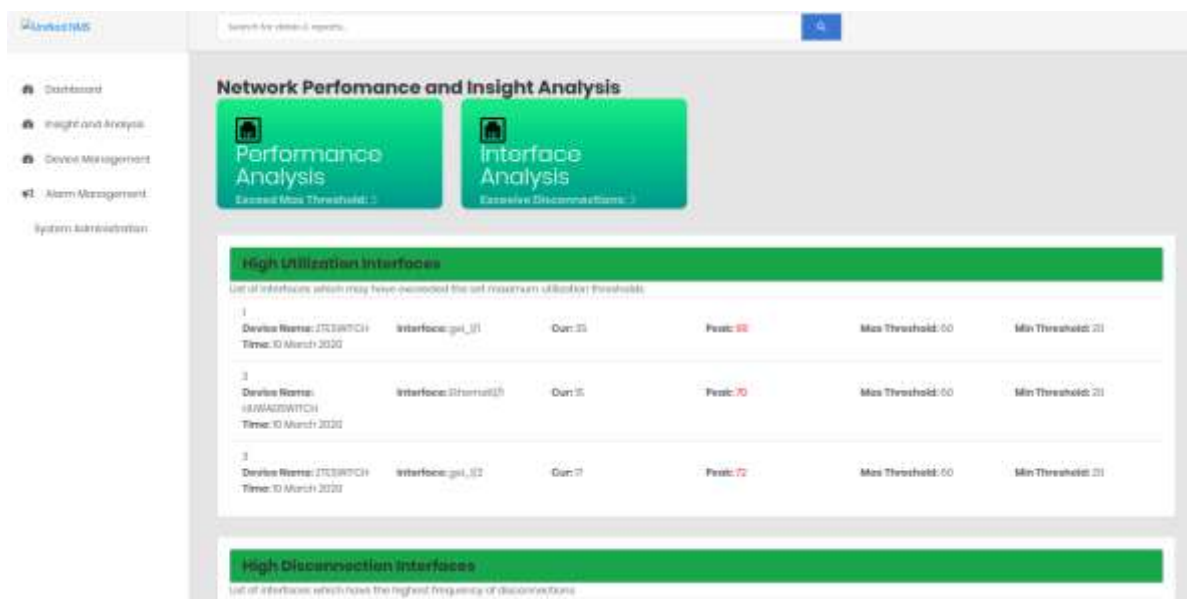


Figure 58: Listing of high utilization interfaces

4.3.15 System properties and performance

This section looks at those system properties whose resource utilization vary with the usage and performance of the software. Of interest is the system CPU performance, system memory utilization and the traffic inflows and outflows from the system server to the various monitored network elements

System CPU information

```
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 4
On-line CPU(s) list: 0-3
Thread(s) per core: 2
Core(s) per socket: 2
Socket(s): 1
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 61
Model name: Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz
Stepping: 4
CPU MHz: 2400.968
CPU max MHz: 3800.0000
CPU min MHz: 800.0000
BogoMIPS: 4789.89
Virtualization: VT-x
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 4096K
NUMA node0 CPU(s): 0-3
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe
syscall nx pdpebgh rdtsdp lm constant tsc arch_perfmon pbs bts rep good nopl xtopology nonstop tsc uperfperf pni pclmulqdq dtes64 monitor d
a_cpl vmx est tm2 asse3 adbg fma cki6 xtrp pdcn pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer amd_xsave aux f16c rdrand lahf_lm ab
n 3dnowprefetch nopl invpcid_single intel_pt sabb ibrs lbrs stibp kaiser tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 sm
ep bmi2 erms invpcid rdtscn adx snap xsaveopt dtherm ida arat pin pts rd_clear flush_l3d
```

Figure 59: CPU information

For the server hardware hosting the system was a 4 core, Core i7 5th generation processor at 2.4GHz was used. The specs were sufficient for our usage as we were handling fewer devices. The database used was sqlite which is very light on resources. For industrial use we would recommend using a more robust database such as Sybase or MySQL and they would be need to deploy this on a separate server or VM for efficiency.

Memory Utilization

The following screen shot show a comparison of the memory usage of the system before and during running the NMS software. From the screenshots it was observed that the application used 540Mbytes of memory, and this memory utilization would obviously increase if more devices are added and more performance indicators are configured for collection. It was noted the system would be ideal for cases with fewer nodes, if more nodes are required the memory needs to be upgraded.

	total	used	free	shared	buff/cache	available
Mem:	16304360	2566756	9660268	1106800	4077336	12238352
Swap:	524284	0	524284			

Figure 60: System memory before starting the application

	total	used	free	shared	buff/cache	available
Mem:	16304360	3102164	9100076	1119432	4102120	11688368
Swap:	524284	0	524284			

Figure 61: System memory when running the application

System Performance Trends



Figure 62: System monitoring when running the application

4.3.16 Consolidating the system

In this section the mechanisms used to consolidate, integrate and combine the three layers making up the different parts of the system are discussed. The following block diagram shows

functional requirements of the entire system divided into three sections.

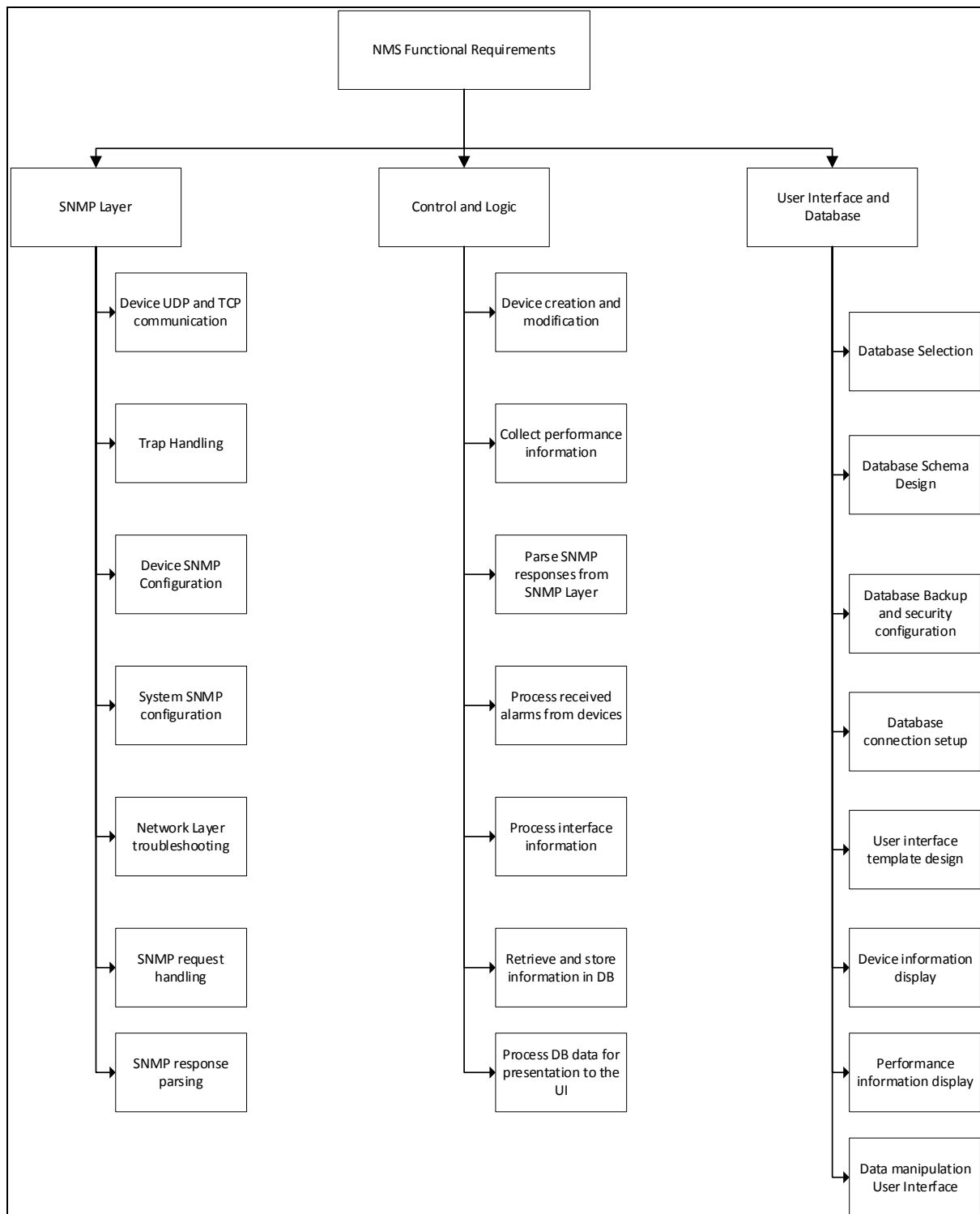


Figure 63: System functional requirements:

From the above functional requirements, it can be shown how the three distinct areas were combined to achieve the task at hand. The control and logic part was responsible for interfacing between the SNMP networking later and the user interface/database. The control logic was integrated with the SNMP layer using the python package `snmp_cmds` and the Django

framework provided database and template framework for integrating with the database and the user interface.

The block diagram below shows the integration points of the system and the technologies used at each interface.

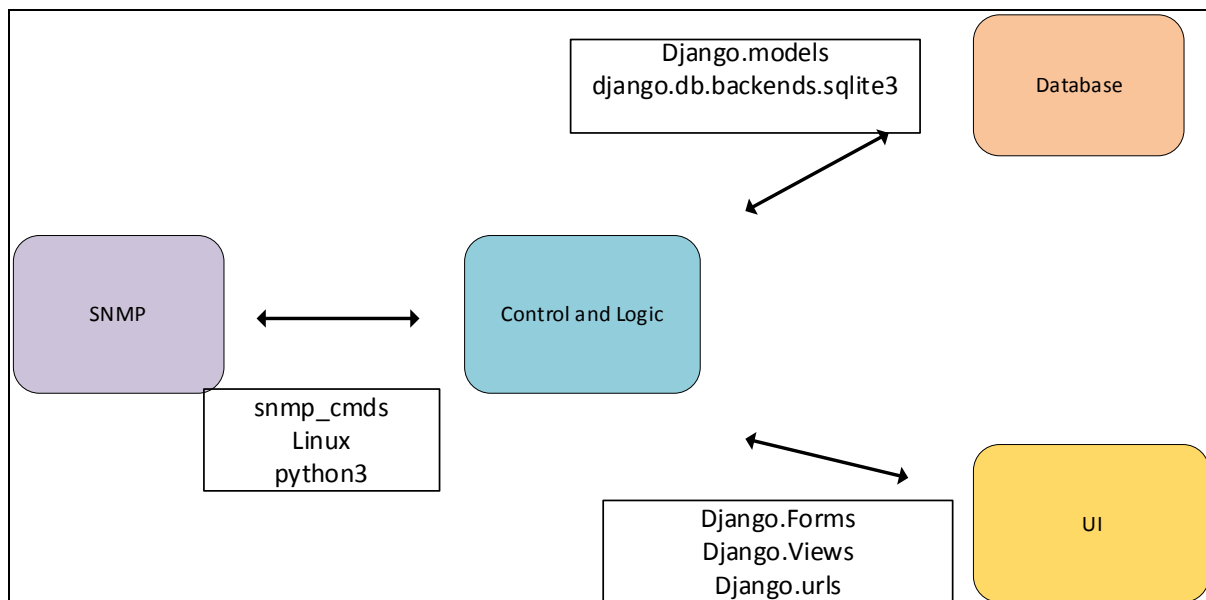


Figure 64: Integration points

SNMP layer to Control layer integration

The interface uses the `snmp_cmds` python package for integrations. The library creates SNMP request from python methods. The return communication from the SNMP tool used basic python variables which were parsed in the control layer for storage and presentation.

Database to Control Layer integration

The interface between the database and the control layer used the Django APIs for establishing communication.

UI to Control Layer integration

The interface between the UI and the control layer used the Django APIs (`Django.models` and `Django.db.backends.sqlite3`) for integration.

4.4 Results Analysis and Discussion

4.1 System Design Performance Evaluation

The trend analysis of the system performance provided a view of the variation of key performance indicators for the system, such as

- The processing capacity utilisation of each respective CPU on the system with time ,action and activity being performed on the system
- The memory utilisation of the system with time and also action or request being processed
- The variation of traffic flow on the network monitoring system with action or request being processed



Figure 65: System KPI Trend Analysis for Interface Performance Request

Start of Performance Query Request

As can be seen in Figure 65, the systems key performance indicators vary considerably with the request being processed and for a request such as interface performance query, this action was shown to be very high demanding on processing power as the CPU utilization rose to as high as 50% on some of the processors

This is mainly due to continuous and repetitive nature of the action performed and also to the number of active or enabled interfaces on the respective network node being monitored. If more nodes were to be connected and monitored, system's server hardware capabilities would need to be upgraded for such actions to be easily undertaken without much delay.

A comparison of the designed system's performance with other industry available network monitoring systems like the OpManager which is one of the few network monitoring systems with multi-vendor monitoring capability and is currently being used by some of the mobile operators, indicates that, high server processing capability is one key performance indicator that has to be fully resourced if more nodes are to be connected and managed by the designed network monitoring system

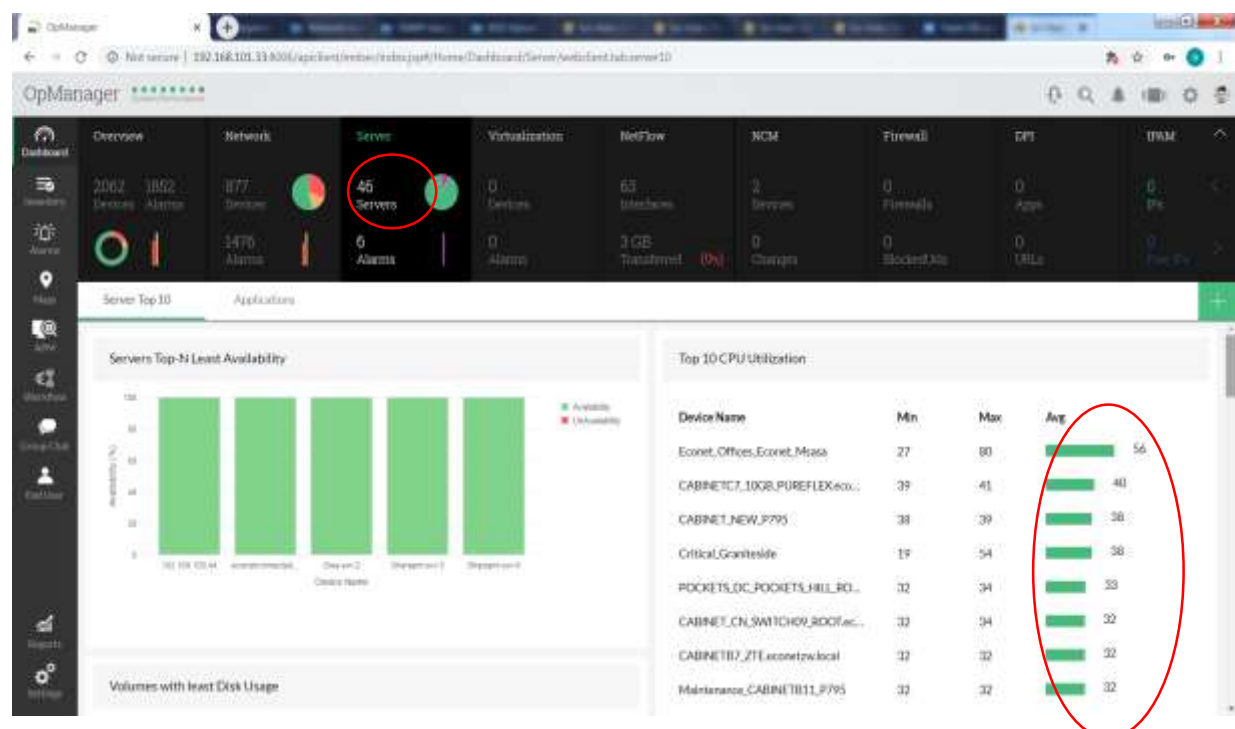


Figure 66: OpManager Server and Node CPU Utilization Capacity

Current industry available network monitoring systems have similar Node CPU utilization performance results for some specific processing requests. A top average of around 50% node CPU Utilization capacity was also noted for OpManager system however this system is even equipped with multiple servers to aid in the multiple processing of requests.

The designed network monitoring system was designed with single server due to cost considerations in acquiring the necessary hardware and software to support a multiple server design.

4.2 System Memory

For the designed Unified Network monitoring system, the memory utilisation for available memory of 15.5 GB was noted to be steady around 20% for the given performance requests that were processed during testing and system operation.



Figure 67: Unified Network Monitoring System Memory Utilisation

This memory utilisation compares very well with that of similar network monitoring systems currently in use in industry, considering that the available memory space of for the design system was 15.5Gigabyte against industry available systems with memory of up to 2Terabytes

and memory utilisation ranging from 10% up to 80% as indicated below. Such seemingly huge hardware capacity requirements are desirable for any system to be able to undertake multiple processes at the same time as well as have the capacity to store , both performance an alarm data for an appreciable amount of time. Typical storage durations for performance and alarm data is at least 3months.

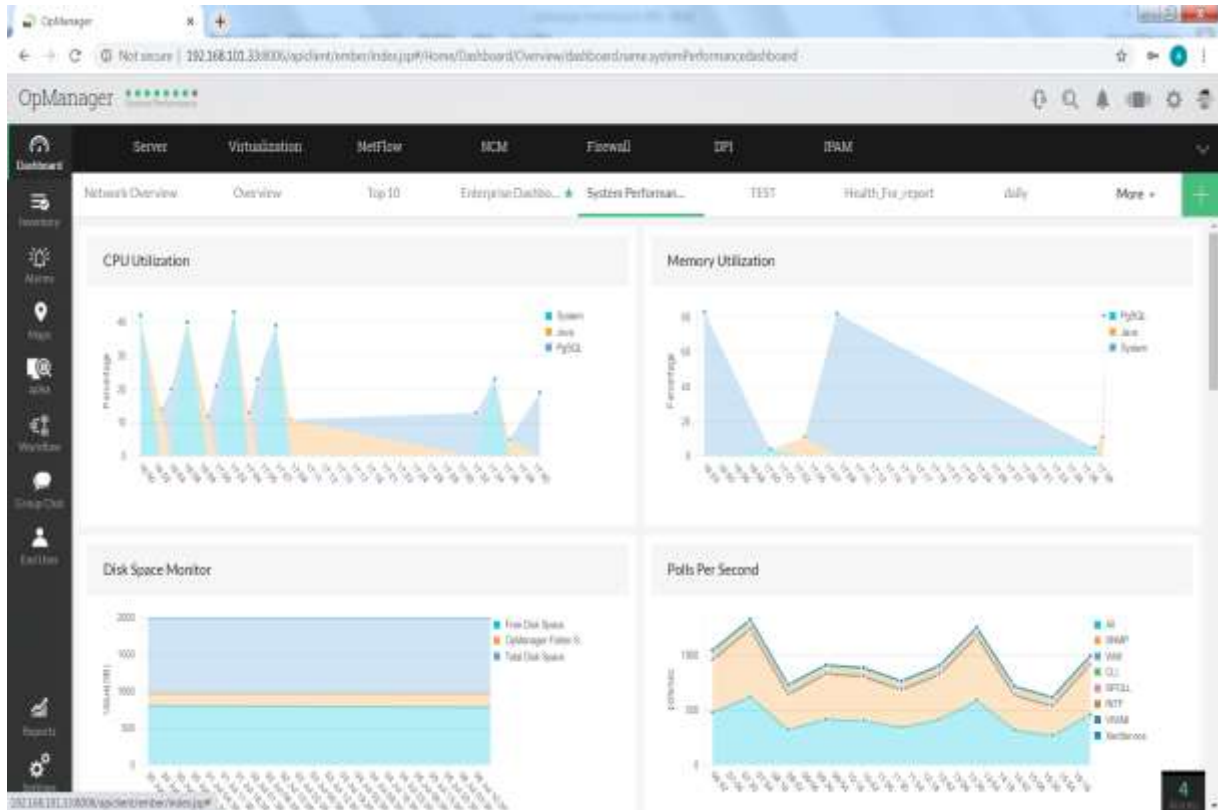


Figure 68: OpManager System Performance

For enterprise deployment of the developed monitoring system, it would be desirable to have more memory capacity to be able to handle memory requests for the multiple requests from the many monitored network nodes.

4.3 System Dashboard

The designed system dashboard and graphical user interface compares fairly well with other industry available network monitoring systems as indicated in the figures below. The designed system dashboard landing webpage has short cut menus to system applications such as, Insight and Analysis for performance management and trends analysis in both graphical and tabular form, Device management for system views of all the devices monitored by the system and for addition of new network nodes that would have been discovered or integrated into network, to enable monitoring from the network monitoring system.

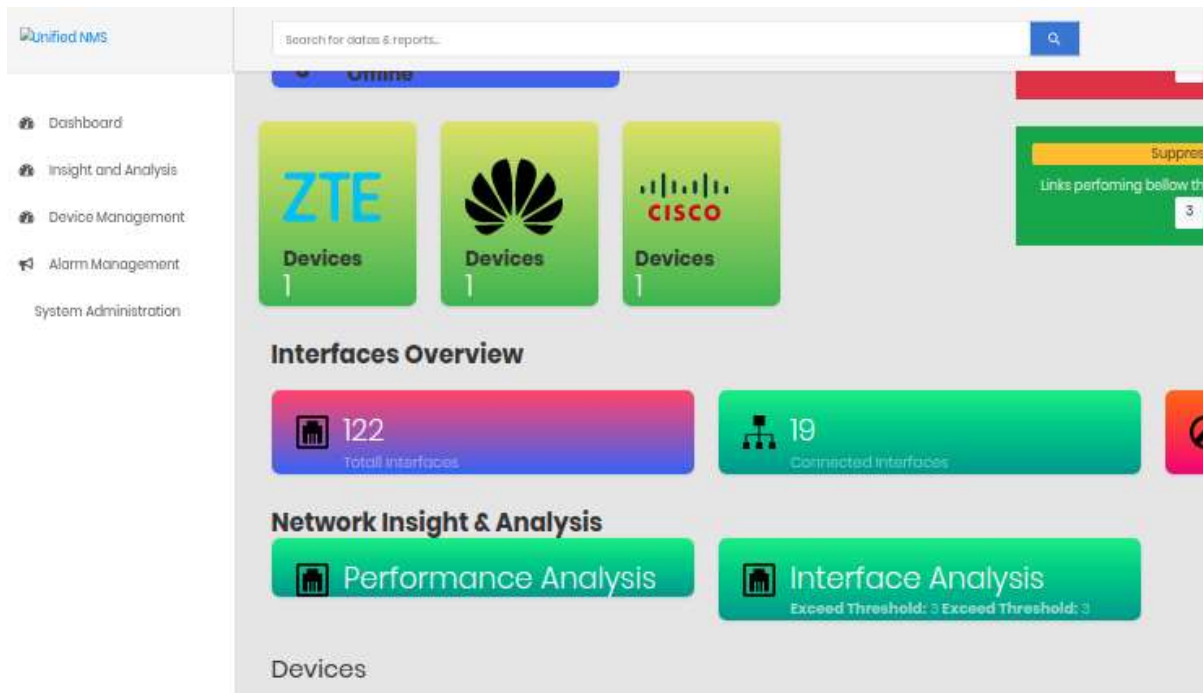


Figure 69: Unified Network Monitoring System Dashboard View

In addition the system dashboard view has tabs for alarm management which provides a system view of current and historical network nodes alarms whilst the System administration tab provides for the updating of new vendor profiles and also system view of all snmp devices in the network.

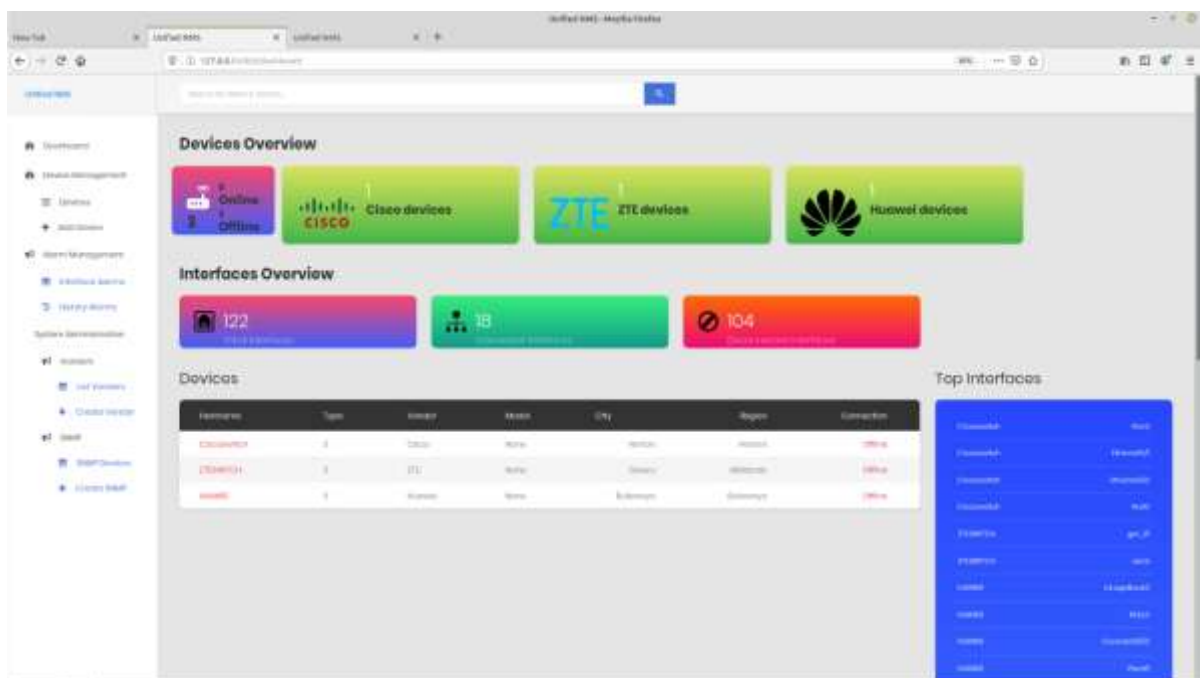


Figure 70: Unified Network Monitoring System Expanded Dashboard View

A comparison of the designed system dashboard view and the dashboard view from similar network monitoring systems already in operation in the industry indicates that though systems currently in use have more features and consequently more items displayed on the dashboard, the basic graphical user interface display items are similar. Links or tabs to alarm management, device management and system administration are available on the designed network monitoring system.

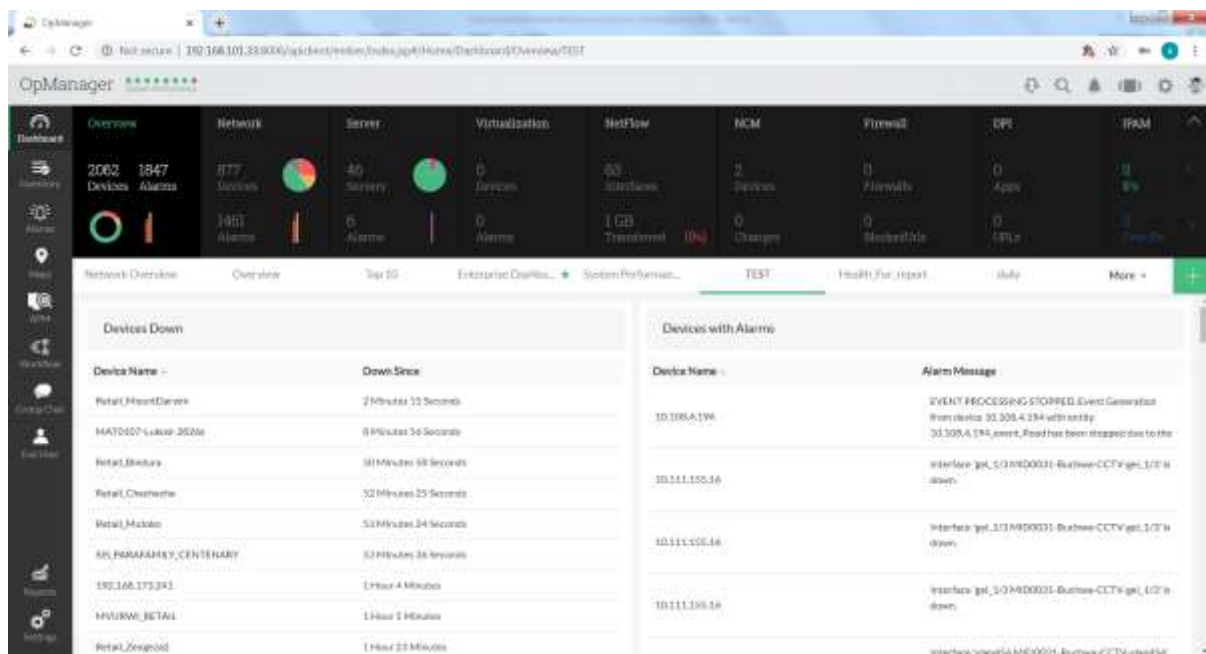


Figure 71: OpManager Graphical User Interface Dashboard View

The other graphical user interface features such as configuration management, inventory management, maps etc. were not included in the development of this monitoring system, purely because of the limited resources that included, time and financial resources to support development of a greater magnitude.

4.4 System Device Templates

Device templates view is a system view that enables one to view the different types of network equipment, their manufacturer or vendor, and also how many of each unique type are currently monitored by the monitoring system. In addition it also provides for quick discovery of new network nodes onto the monitoring system if similar nodes from a manufacturer are already managed by the monitoring system

Name	Vendor Index	Models	Devices
Cisco	8	0	1
Huawei	208	0	1
ZTE	3802	0	1

Figure 72: Unified Network Monitoring System Vendor List View

The network monitoring system was designed to be a multi-vendor solution and as indicated in the diagram above, three vendors' network elements, namely a ZTE switch, a Huawei switch and a Cisco switch were successfully integrated and monitored on the system. This capability compares equally with similar capabilities from other network monitoring systems currently in use in the industry, if the view from the OpManager below is anything to go by.

Name	Category	Devices	Actions
Basic Firewall	Firewall	1	[Edit] [Delete] [Add]
ZTE 1884-DC	Switch	1	[Edit] [Delete] [Add]
ZTE PRAN 9000 SE	Router	1	[Edit] [Delete] [Add]
Catalyst Switch	Switch	1	[Edit] [Delete] [Add]
Check Point Firewall	Firewall	1	[Edit] [Delete] [Add]
Cisco 3600 Series	Router	1	[Edit] [Delete] [Add]
Cisco 3640	Router	1	[Edit] [Delete] [Add]
Cisco 5000 WLC	Wireless	1	[Edit] [Delete] [Add]
Cisco AIR-CT5512	Wireless	1	[Edit] [Delete] [Add]
Cisco Catalyst 1900	Switch	1	[Edit] [Delete] [Add]
Cisco Catalyst 4912G	Switch	1	[Edit] [Delete] [Add]
Cisco Catalyst 4948E-F Switch	Switch	1	[Edit] [Delete] [Add]

Figure 73: OpManager Network Monitoring System Device Templates View

4.5 Network Elements Interface Utilization Measurement

The ability to measure or trend interface utilisation is an important key performance indicator in that it provides a predictive capability of the network's demands allowing for futuristic planning necessary to guarantee or maintain good quality of service delivery. This feature was incorporated in the designed system solution under the Network Insights and Analysis tab. Sample results from one such test are as shown below



Figure 74: Unified Network Monitoring System Interface Utilisation on Cisco Switch

The feature and functionality compares fairly well with similar features on other similar monitoring systems even though there is room for further enhancements to match or surpass the current systems. The designed functionality provides for the measurement and trending of an interface's utilisation over a set period, whilst the feature on some mature monitoring systems includes as well the ability to display this data with even more detail such displaying outgoing and incoming traffic utilisation, as indicated below.

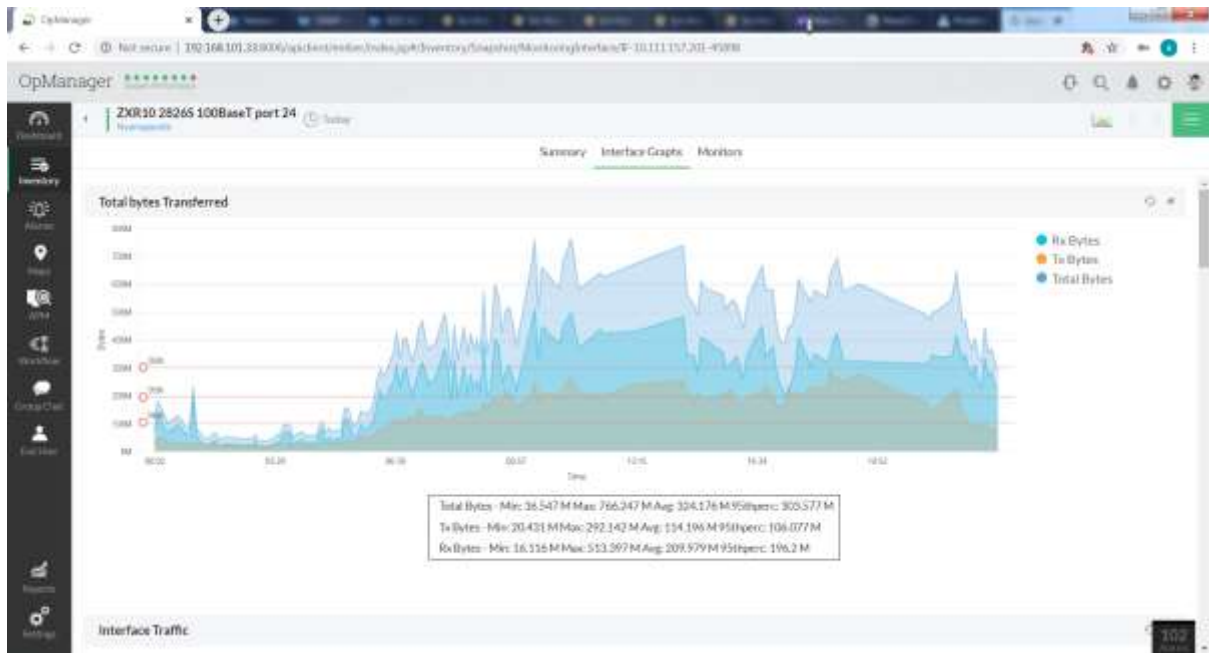


Figure 75: OpManager ZTE Switch Interface Utilisation

4.6 Interface Alarms Monitoring

The ability to have an alarm interface status view for all the interfaces on a network element is important especially during failure of some but not all services on a network element. This functionality allows for the operation and maintenance teams to quickly identify those interfaces that would be faulty or down and quickly narrow down the possible faulty ones.

Device	Index	Description	Status	Occur Time	Close Time
uwrng-gw	1	(Ethernet/3)	Connected	None	Jan 30, 2020 10:00 pm
uwrng-gw	2	(Ethernet/3)	Connected	None	Jan 30, 2020 10:00 pm
uwrng-gw	3	(Ethernet/3)	Disconnected	Jan 30, 2020 10:20 pm	None
uwrng-gw	4	(Ethernet/3)	Disconnected	Jan 30, 2020 10:21 pm	None
uwrng-gw	5	Serail/3	Disconnected	Jan 30, 2020 10:3 pm	None
uwrng-gw	6	Serail/3	Disconnected	Jan 30, 2020 10:3 pm	None
uwrng-gw	7	Serail/3	Disconnected	Jan 30, 2020 10:3 pm	None
uwrng-gw	8	Serail/3	Disconnected	Jan 30, 2020 10:3 pm	None

Figure 75: Interface Alarm Status View

This capability was incorporated in the designed system as indicated in the diagram above showing the status and type of interfaces on one network element. A comparison of this

capability with other network monitoring systems available on the market indicates that the basic functions for this feature which are, interface status and type of interface, are quite comparable. Enhancements to incorporate functionalities such as interface protocol type or cumulative traffic on any interface were not explored purely for reasons of time management during this research design.

Status	Interface Name	Type	Rx Traffic	Tx Traffic	Action
Clear	null-eth0	Other	0 (0.0%)	0 (0.0%)	
Clear	ge0-ge1	Ethernet	0 (0.0%)	0 (0.0%)	
Clear	pos3-0/0/0/1 To_MAT0008-Spring Range-WGE_pos3-0/0/0/1-pos3-0/0/0/1	PPP	30.696K (19.74%)	31.222K (30.08%)	
Clear	pos3-0/0/0/2 To_MAT0008-Spring Range-WGE_pos3-0/0/0/2-pos3-0/0/0/2	PPP	11.159M (71.08%)	30.093M (19.32%)	
Clear	pos3-0/0/0/3 To_MAT0008-Spring Range-WGE_pos3-0/0/0/3-pos3-0/0/0/3	PPP	17.303M (11.13%)	30.941M (19.65%)	
Clear	pos3-0/0/0/4 To_MAT0008-Spring Range-WGE_pos3-0/0/0/4-pos3-0/0/0/4	PPP	2.782M (5.0%)	34.13M (19.37%)	
Trouble	pos3-0/0/0/5-pos3-0/0/0/5	PPP	0 (0.0%)	0 (0.0%)	
Trouble	pos3-0/0/0/6-pos3-0/0/0/6	PPP	0 (0.0%)	0 (0.0%)	
Trouble	pos3-0/0/0/7-pos3-0/0/0/7	PPP	0 (0.0%)	0 (0.0%)	
Trouble	pos3-0/0/0/8-pos3-0/0/0/8	PPP	0 (0.0%)	0 (0.0%)	

Figure 76: Device Monitoring Snapshot

4.7 Comparison of results with other related works

Table 15: Comparison of results

Authors	Artwell Magadzire 2020	C.C.Li, Z.S.Ji, Feng Wang, P Wang, Y Wang, Z.C.Zhang 2016	Vincent Geddes 2008	
Title	<i>“Unified Multi-Vendor Network Monitoring System Development”</i>	<i>“The Network Monitoring System based on CACTI for EAST”</i>	<i>“Design and Implementation of a Scalable Network Monitoring System”</i>	COMMENTS
System Stability as a factor of Operating System used	Linux Ubuntu Operating System		Deamon processes to run on Debian GNU/Linux Operating Systems	For stable and robust systems, linux operating systems are the predominant preference
System design architecture	Web-server online monitoring system using Django network framework based on the B/S architecture	Web-server online monitoring system using Cacti Framework based on the B/S architecture	Web-server monitoring system implemented using the SCGI protocol	Most network monitoring systems use B/S architecture for the flexibility that it provides
System design methodology	Three distinct paradigms. Emphasis was to allow for continuous system development without the whole system being shut down		Iterative with more and more functionality progressively developed. Emphasis on the system being buildable and runnable at all times.	Continuous system development without any downtime being experienced is key
Protocol and Databases	SNMP for devices and HTTP for GUI interface. SQLite3 and Redis databases	SNMP for NE devices and HTTP for GUI interface. RRD and MySQL databases	SCGI protocol for web-service interface, HTTP for clients, binary protocol and CLI utility for agents interface	The ease brought about by SNMP protocol in data collection enhanced development

Authors	Artwell Magadzire 2020	C.C.Li, Z.S.Ji, Feng Wang, P Wang, Y Wang, Z.C.Zhang 2016	Vincent Geddes 2008	
Programming Language	Python Programming Language	PHP Programming Language	D Programming Language	Programming language with a big community provides for wider support base
System Feature Activations and Licensing	No license requirements for all the features supported by the system	Open source software used for system development	Software Licensing implementation under the terms of the MIT/X11 license.	This research's design provides for a cost effective solution to network monitoring of nodes in heterogeneous transmission networks

Highlights of some of the key findings and comparisons by previous researchers on the subject of network monitoring systems design are captured in the table above. The system design architectures were observed to be almost similar in most cases, due to the flexibility aspect that comes with web server systems.

Different approaches were taken to address the system design methodology but the predominant factor in all the previous research works' methodology, was the need for continuous system development without incurring any system downtime.

In conclusion, it can thus be observed that different approaches can and have been used to successfully address network monitoring challenges in a multi-vendor environment. The cost associated with implementing and subsequent maintenance of the system later, for a particular solution stands out as one of the key factors to consider in selecting the best solution with feature activations and licensing impacting hugely on this consideration.

CHAPTER FIVE: CONCLUSION AND RECOMMENDATIONS

The primary outcome of this research has been to provide a conceptual framework of methods and tools that can be implemented to monitor diverse mobile communications network elements. The research focussed on the design and development of a network monitoring system which is not vendor specific and could manage diverse transmission mobile network elements with particular emphasis on the monitoring server framework design. The network monitoring system was supposed to be capable of monitoring multi-vendor network elements, notably ZTE, Huawei and Cisco devices, in addition to carrying out performance measurement and output service alarms of these devices on to a dashboard without the need for licenses for these feature activations and capabilities. The designed system achieved these objectives.

A web-server network monitoring system based on the Django network framework was designed. Three IP switching transmission network elements from ZTE, Huawei and Cisco were used to verify and prove the feature and functional capabilities of the designed monitoring system. This was done by connecting them in a network setup and confirming the status of the connected network elements from the network monitoring system.

The designed system had basic functional capabilities such fault management and performance management. Based on the results it can be concluded that a multi-vendor network monitoring system design that does not require a lot of OPEX and CAPEX for mobile operators, with the desired features and functionalities is feasible.

This research work has been used as part of a larger project. It forms the middle level of the complete unified network monitoring system. The physical interface and user interface component layers depend on this research for a reliable and high-quality network monitoring system as it completes the configuration and integration of all aspects of the system such as the display mechanisms for the graphical user interface and SNMP traps handling.

From the results it was demonstrated that the designed network monitoring system had basic functional system capabilities such as alarm monitoring and interface utilization. Enhanced feature and functional capabilities, such as early warning capability or service quality monitoring, for the network monitoring system would require further research work on the contents and structure of MIB files of each respective vendor equipment. This will be necessary to be able to manipulate the MIB files and hence be able to have both read and write rights to the respective network elements. This is one possible area of further research, the

implementation of configuration management capabilities of multi-vendor network elements from one network monitoring system.

Based on the results, it is recommended that further research work be explored in the area of upgrading multi-vendor network monitoring systems to multi-vendor network management systems

REFERENCES

- [1] ITU-T Recommendation M.3010, Principles for a Telecommunication Management Network , May 1996.
- [2] Ludlow, G. (1997). Network management. IEE Colloquium on How to Compete and Connect: Understanding the Engineering of Telecommunications Network Interconnection.
- [2] Abubucker Samsudeen Shaffi, Mohanned Al-Obaidy. MANAGING NETWORK COMPONENTS USING SNMP. International Journal of Scientific Knowledge Computing and Information Technology. April 2013. Vol. 2, No.3
- [3] S Sasidharan, “Efficient Network Monitoring in Enterprise Data Networks Using Node Capabilities in Computer Network and Multimedia Technology” CNMT 2009.International Symposium on, pp. 1-4 ,2009
- [4]Case, J., Fedor, M., Schoffstall, M., Davin, J. 1990. “A Simple Network Management Protocol (SNMP)” Request for Comments 1157, Network Working Group. Accessed 22 January, 2020. <http://tools.ietf.org/rfc/rfc1157.txt>
- [5]Van Renesse, R., Birman, K. P., Vogels, W. 2003. “Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining.” ACM Trans. Comput. Syst. Vol. 21.2 (May. 2003), p. 164–206. [doi:10.1145/762483.762485](https://doi.org/10.1145/762483.762485)
- [6] Sottile, M., Minnich, R. 2002. “Supermon: A high-speed cluster monitoring system” In Proceedings of Cluster 2002 [doi:10.1109/CLUSTER.2002.1137727](https://doi.org/10.1109/CLUSTER.2002.1137727)
- [7] Massie, M. L., Chun, B. N. & Culler, D. E. 2004. “The ganglia distributed monitoring system: design, implementation, and experience” Parallel Computing, Vol. 30.7, p. 817–840. [doi:10.1016/j.parco.2004.04.001](https://doi.org/10.1016/j.parco.2004.04.001)
- [8] Xianmin Wei. Design and Implementation of Network Management System Based on Mixed-mode.2012 International Conference on Applied Physics and Industrial Engineering pp871-876

- [9] Yongqi Han, Taihao Li, Yun Zhang, Liying Cao. Research of Network Monitoring Based on SNMP: 2013 Third International Conference on Instrumentation, Measurement, Computer, Communication and Control pg 1-4
- [10] Regis J, “BUD” Bates. Broadband Telecommunications Handbook 2nd Edition. McGraw-Hill TELECOM PROFESSIONAL
- [11] Al Kovalick, “Systems Management and Monitoring ,” Video Systems in an IT Environment (Second Edition), 2009, Pages 345-372.
- [12] Suryadiputra Liawatimena, Edi Abduran, Ford Lumban Gaol, Harco Leslie Hendric Spits Warnars, Benfano Soewito, Bahtiar Saleh Abbas, Agung Trisetyarso, Antoni Wibowo. Django Web Framework Software Metrics Measurement Using Radon and Pylint. The 1st 2018 INAPR International Conference, 7 Sept 2018, Jakarta, Indonesia
- [13] D. P. Pop and A. Altar. (2014). Designing an MVC model for rapid web application development. Procedia Engineering, 69, 1172-1179.
- [14] M. Aniche, G. Bavota, C. Treude, M. A. Gerosa, and A. van Deursen, “Code smells for Model-View-Controller architectures,” Empir. Softw. Eng., vol. 23, no. 4, pp. 2121–2157, 2018.
- [15] Vincent Geddes, Greg Kempe, Michelle M Kuttel, Patrick Marais 2020.”Design and Implementation of a Scalable Network Monitoring System” University of Cape Town.
<https://www.researchgate.net/publication/255669527>
- [16] C.C. Li, Z.S. Ji, Feng Wang, P Wang, Y Wang, Z.C.Zhang 2016.” The Network Monitoring System based on CACTI for EAST” 2016 IEEE-NPSS Real Time Conference (RT)
- [17] Song, C., Huo, R., Wang, S., & Lv, C. (2019). Transformer Equipment Temperature Monitoring Based on the Network Framework of Django. 2019 Chinese Automation Congress (CAC).
- [18] Xue Yaowei. Design and implementation of automatic generation module based on Django framework management interface [D]. Harbin: Harbin Institute of Technology, 201
- [19] POTRAZ “POTRAZ Abridged Postal and Telecommunications Sector Performance Report: First Quarter 2020” [Online] Available:
http://www.potraz.gov.zw/?page_id=527 [Accessed June2020]

<https://docs.djangoproject.com/en/3.0/>

APPENDICES

Project Django Settings

```
"""
Django settings for NMS project.

Generated by 'django-admin startproject' using Django 2.2.4.

For more information on this file, see
https://docs.djangoproject.com/en/2.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/2.2/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/2.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '9@-uznt97ct18)@@5o5ab^k@rxac47&c#xbi0c@gpjy95&njub'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ['192.168.1.1', 'localhost', '127.0.0.1']

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'uNMS',
    'widget_tweaks',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'NMS.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

```

    },
]

WSGI_APPLICATION = 'NMS.wsgi.application'

# Database
# https://docs.djangoproject.com/en/2.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

# Password validation
# https://docs.djangoproject.com/en/2.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/2.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.2/howto/static-files/

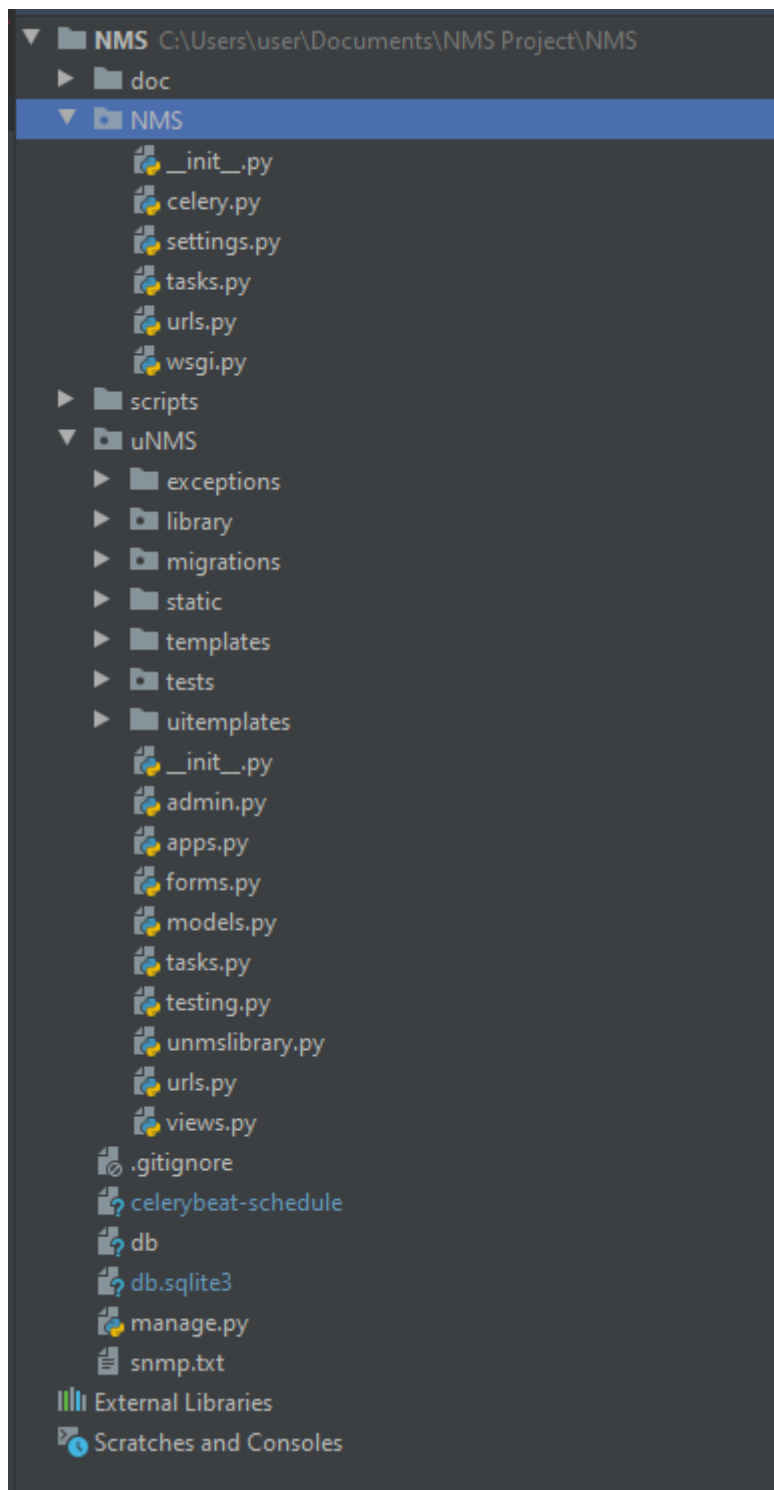
STATIC_URL = '/static/'
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)

# Celery application configurations
CELERY_BROKER_URL = 'redis://localhost:6379/0'
CELERY_RESULT_BACKEND = 'redis://localhost:6379/0'
CELERY_ACCEPT_CONTENT = ['application/json']
CELERY_RESULT_SERIALIZER = 'json'
CELERY_TASK_SERIALIZER = 'json'
CELERY_BEAT_SCHEDULE = {
    'get-device-heartbeat-every-10-seconds': {
        'task': 'uNMS.tasks.check_all_device_heartbeat',
        'schedule': 10.0,
    },
    'update-device-interface-every-30-seconds': {
        'task': 'uNMS.tasks.update_all_device_int',
        'schedule': 30.0,
    },
}







```

```
    },  
    'update_all_device_performance': {  
        'task': 'uNMS.tasks.update_all_device_performance',  
        'schedule': 15.0,  
    }  
}  
CELERY_TIMEZONE = 'Africa/Harare'
```

Project File Structure



Project Script Files

 commands	2020-03-05 02:59 PM	File	1 KB
 snmp	2020-03-05 02:59 PM	Text Document	62 KB
 snmptrapd	2020-03-05 02:59 PM	Python File	2 KB
 startcelery	2020-03-05 02:59 PM	File	1 KB
 startnms	2020-03-05 02:59 PM	File	2 KB
 stopnms	2020-03-05 02:59 PM	File	1 KB

Project Forms Definitions

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from .models import *

class CreateVendorForm(forms.ModelForm):
    """
    Form to create a bot.
    """
    class Meta:
        model = Vendor
        fields = ('name', 'vendor_index')

class CreateModelForm(forms.ModelForm):
    class Meta:
        model = Model
        fields = ('name', 'vendor')

class CreateDeviceForm(forms.ModelForm):
    class Meta:
        model = Device
        fields = ('ipaddress', 'type', 'city', 'region')

class CreateSNMPProfileForm(forms.ModelForm):
    class Meta:
        model = SNMPdetails
        fields = ('device', 'rcommunity', 'wcommunity', 'version', 'timeout', 'retries', 'security_level')

class UpdateSNMPProfileForm(forms.ModelForm):
    class Meta:
        model = SNMPdetails
        fields = ('rcommunity', 'wcommunity', 'version', 'timeout', 'retries', 'security_level')

class InterfaceMonitorConfigForm(forms.ModelForm):
    class Meta:
        model = InterfacePerformance
        fields = ('device', 'interface', 'utilization', 'inoctates', 'outoctates')

class UpdateInterfaceMonitorConfigForm(forms.ModelForm):
    class Meta:
        model = InterfacePerformance
        fields = ('utilization', 'inoctates', 'outoctates')
```


Project Views Definitions

```
# Create your views here.
class CreateVendor(CreateView):
    template_name = 'uNMS/basic_create.html'
    form_class = CreateVendorForm
    success_url = reverse_lazy('uNMS:list_vendors')

class ListVendor(ListView):
    model = Vendor
    paginate_by = 20
    template_name = 'uNMS/vendor_list.html'
    context_object_name = 'vendors'

class DetailVendor(DetailView):

    model = Vendor
    template_name = 'uNMS/vendor_list.html'
    fields=('name','vendor')
    context_object_name = 'vendor'

class UpdateVendor(UpdateView):
    template_name = 'uNMS/basic_create.html'
    form_class = CreateVendorForm
    model = Vendor
    success_url = reverse_lazy('uNMS:list_vendors')

class DeleteVendor(DeleteView):
    model = Vendor
    success_url = reverse_lazy('uNMS:list_vendors')

class CreateModel(CreateView):
    template_name = 'uNMS/basic_create.html'
    form_class = CreateModelForm
    success_url = reverse_lazy('uNMS:list_vendors')

#####Device Views#####

class CreateDevice(CreateView):
    template_name = 'uNMS/basic_create.html'
    form_class = CreateDeviceForm
    success_url = reverse_lazy('uNMS:list_device')

class UpdateDevice(UpdateView):
    template_name = 'uNMS/basic_create.html'
    form_class = CreateDeviceForm
    model = Device
    success_url = reverse_lazy('uNMS:list_device')

class DeviceList(ListView):
    model = Device
    paginate_by = 10
    template_name = 'uNMS/device_list.html'
    context_object_name = 'devices'
```

Models Definitions

```
class Vendor(models.Model):

    name=models.CharField(max_length=200)
    vendor_index=models.IntegerField()

    @property
    def models_count(self):
        """
        Return the count of models associated with the vendor
        :return:
        """
        models = self.models()
        return models.count()

    def models(self):
        """
        get all models related to this vendor
        :return: Models List
        """
        models = Model.objects.filter(vendor=self)
        return models

    def devices(self):
        """
        Get devices associated with the Vendor.
        :return:
        """
        devices = Device.objects.filter(enterprise_id = self.vendor_index)
        return devices

    def device_count(self):
        """
        Get device count for devices related to the vendor
        :return:
        """
        devicelist = self.devices()

        return devicelist.count()

    def __str__(self):
        return self.name

class Model(models.Model):
    """
    Device model database table
    """
    name=models.CharField(max_length=200,)
    vendor=models.ForeignKey(Vendor,on_delete=models.CASCADE)
    model_index = models.IntegerField(null=True)

    def __str__(self):
        name = self.name
        if name is None:
            return 'Unconfigured'
        return name

class Device(models.Model):

    DEVICE_TYPE = (('R','Router'),
                   ('S','Switch'))
    VENDORS = (('HW','Huawei'),
               ('CS','Cisco'),
               ('ZT','ZTE'))

    ipaddress = models.CharField(max_length=15,null=True)
    type = models.CharField(choices=DEVICE_TYPE,max_length=20)
    model = models.ForeignKey(Model,on_delete=models.CASCADE,null=True)
    city= models.CharField(max_length=200,null=True)
    region= models.CharField(max_length=200,null=True)
    version = models.CharField(max_length=200,null=True)
```

```

date_created = models.DateTimeField(auto_now_add=True)
online = models.BooleanField(default=False)
hostname = models.CharField(max_length=200,null=True)
enterprise_id = models.CharField(max_length=200,null=True)

def __str__(self):
    if self.hostname is None:
        return self.ipaddress
    else: return self.hostname

def get_absolute_url(self):
    return reverse('uNMS:update_device', kwargs={'pk': self.pk})

@property
def enterprise_name(self):
    """
    Property to return the vendor name of the device
    :return:
    """
    vendor = Vendor.objects.get(vendor_index=self.enterprise_id)
    return vendor.name

@property
def model_name(self):
    """
    Property to return the model name of the device
    :return:
    """
    pass

@property
def has_snmp(self):
    """
    Check if device has SNMP details configured on it
    :return: Boolean
    """
    try :
        snmp_profile = SNMPdetails.objects.get(device = self)
        if snmp_profile :
            return True
    except :
        return False

class Alarm(models.Model):

    occur_time = models.DateTimeField(null=True,auto_now=timezone)
    clear_time = models.DateTimeField(null=True,auto_now=timezone)
    device = models.ForeignKey(Device,editable=True,on_delete=models.CASCADE)
    name = models.CharField(max_length=200) # type: CharField
    board = models.CharField(max_length=200)

class Interface(models.Model):

    device = models.ForeignKey(Device, null=False,on_delete=models.CASCADE)
    device_ip = models.CharField(max_length=200,null=True)
    ifType = models.CharField(max_length=200,null=True)
    ifIndex= models.IntegerField(null=False,blank=False)
    ifDescription = models.CharField(max_length=200, null=True)
    ifPhysAddress = models.CharField(max_length=200, null=True)
    ifAdminStatus = models.CharField(max_length=10, null=True)
    ifOperStatus = models.BooleanField(default=False)
    ifSpeed = models.IntegerField(null=True)
    ifMtu = models.IntegerField(null=True)

    class Meta:
        # unique_together = ('device','ifIndex')
        # constraints = [
        #     models.UniqueConstraint(fields=['device','ifIndex'],name='unique_device_interface')
        # ]

```

```

        indexes = [
            models.Index(fields=['device','ifIndex'],)
        ]

    def __str__(self):
        return self.ifDescription

class InterfaceTrap(models.Model):
    """
    Table for interface trap update
    """
    occur_time = models.DateTimeField(null=True)
    clear_time = models.DateTimeField(null=True)
    interface = models.OneToOneField(Interface, editable=True,
on_delete=models.CASCADE,primary_key=True)
    linkStatus = models.BooleanField(default=True)
    name = models.CharField(max_length=200)

class InterfacePerformance(models.Model):
    """
    A model to store interface related performance monitoring
    """
    interface = models.ForeignKey(Interface,on_delete=models.CASCADE)
    utilization = models.BooleanField(default=False) # set true if interface is to monitor utilisation
    inoctates = models.BooleanField(default=False) # set True if interface is to monitor in octates
    outoctates = models.BooleanField(default=False) # set true if interface is to monitor out octates
    device = models.ForeignKey(Device,on_delete=models.CASCADE)

    class Meta:
        constraints = [
            models.UniqueConstraint(fields=['interface','device'], name= 'unique_interface')
        ]

class InterfaceIndicatorCounter(models.Model):
    """
    Base model for all interface indicators. All indicators inherit this class
    """
    time = models.DateTimeField(null=False)
    value = models.IntegerField(null = True)

class InterfaceUtilization(models.Model):
    """
    A model to store interface utilization parameters
    """
    interface = models.ForeignKey(Interface,on_delete=models.CASCADE)
    value = models.FloatField(null=True)
    time = models.DateTimeField(null=False)

class InterfaceInboundOctates(InterfaceIndicatorCounter):
    """
    A model to store interface inbound octates
    """
    interface = models.ForeignKey(Interface, on_delete=models.CASCADE)

class InterfaceOutboundOctates(InterfaceIndicatorCounter):
    """
    A model to store the interface outbound octates
    """
    interface = models.ForeignKey(Interface, on_delete=models.CASCADE)

class SNMPdetails(models.Model):
    VERSIONS= ((1,1),
               (2,2),

```

```

        (3,3),)

SECURITY=(( 'no', 'no_auth_or_privacy'),
            ('auth', 'auth_without_privacy'),
            ('ap', 'auth_with_privacy'))

device = models.OneToOneField(Device, on_delete=models.CASCADE, primary_key=True)
rcommunity = models.CharField(max_length=200)
wcommunity = models.CharField(max_length=200)
version = models.IntegerField(choices=VERSIONS, default=1)
timeout = models.IntegerField(default=5)
retries = models.IntegerField(default=5)
security_level = models.CharField(choices=SECURITY, max_length=3, default='no')

def get_absolute_url(self):
    return reverse('uNMS:detail_snmpprofile', kwargs={'pk': self.pk})

```

Database Schema

Name	Type	Schema
auth_group		CREATE TABLE "auth_group" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(150) NOT NULL UNIQUE)
id	integer	"id" integer NOT NULL
name	varchar(150)	"name" varchar(150) NOT NULL UNIQUE
auth_group_permissions		CREATE TABLE "auth_group_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "group_id" integer NOT NULL REFERENCES "auth_group" ("id") DEFERRABLE INITIALLY DEFERRED, "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
group_id	integer	"group_id" integer NOT NULL
permission_id	integer	"permission_id" integer NOT NULL
auth_permission		CREATE TABLE "auth_permission" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "content_type_id" integer NOT NULL REFERENCES "django_content_type" ("id") DEFERRABLE INITIALLY DEFERRED, "codename" varchar(100) NOT NULL, "name" varchar(255) NOT NULL)
id	integer	"id" integer NOT NULL
content_type_id	integer	"content_type_id" integer NOT NULL
codename	varchar(100)	"codename" varchar(100) NOT NULL
name	varchar(255)	"name" varchar(255) NOT NULL
auth_user		CREATE TABLE "auth_user" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "password" varchar(128) NOT NULL, "last_login" datetime NULL, "is_superuser" bool NOT NULL, "username" varchar(150) NOT NULL UNIQUE, "first_name" varchar(30) NOT NULL, "email" varchar(254) NOT NULL, "is_staff" bool NOT NULL, "is_active" bool NOT NULL, "date_joined" datetime NOT NULL, "last_name" varchar(150) NOT NULL)
id	integer	"id" integer NOT NULL
password	varchar(128)	"password" varchar(128) NOT NULL
last_login	datetime	"last_login" datetime
is_superuser	bool	"is_superuser" bool NOT NULL
username	varchar(150)	"username" varchar(150) NOT NULL UNIQUE
first_name	varchar(30)	"first_name" varchar(30) NOT NULL
email	varchar(254)	"email" varchar(254) NOT NULL
is_staff	bool	"is_staff" bool NOT NULL
is_active	bool	"is_active" bool NOT NULL
date_joined	datetime	"date_joined" datetime NOT NULL
last_name	varchar(150)	"last_name" varchar(150) NOT NULL
auth_user_groups		CREATE TABLE "auth_user_groups" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "group_id" integer NOT NULL REFERENCES "auth_group" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL

name	type	Schema
user_id	integer	"user_id" integer NOT NULL
group_id	integer	"group_id" integer NOT NULL
auth_user_user_permissions		CREATE TABLE "auth_user_user_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
user_id	integer	"user_id" integer NOT NULL
permission_id	integer	"permission_id" integer NOT NULL
django_admin_log		CREATE TABLE "django_admin_log" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "action_time" datetime NOT NULL, "object_id" text NULL, "object_repr" varchar(200) NOT NULL, "change_message" text NOT NULL, "content_type_id" integer NULL REFERENCES "django_content_type" ("id") DEFERRABLE INITIALLY DEFERRED, "user_id" integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED, "action_flag" smallint unsigned NOT NULL CHECK ("action_flag" >= 0))
id	integer	"id" integer NOT NULL
action_time	datetime	"action_time" datetime NOT NULL
object_id	text	"object_id" text
object_repr	varchar(200)	"object_repr" varchar(200) NOT NULL
change_message	text	"change_message" text NOT NULL
content_type_id	integer	"content_type_id" integer
user_id	integer	"user_id" integer NOT NULL
action_flag	smallint unsigned	"action_flag" smallint unsigned NOT NULL CHECK("action_flag" >= 0)
django_content_type		CREATE TABLE "django_content_type" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app_label" varchar(100) NOT NULL, "model" varchar(100) NOT NULL)
id	integer	"id" integer NOT NULL
app_label	varchar(100)	"app_label" varchar(100) NOT NULL
model	varchar(100)	"model" varchar(100) NOT NULL
django_migrations		CREATE TABLE "django_migrations" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app" varchar(255) NOT NULL, "name" varchar(255) NOT NULL, "applied" datetime NOT NULL)
id	integer	"id" integer NOT NULL
app	varchar(255)	"app" varchar(255) NOT NULL
name	varchar(255)	"name" varchar(255) NOT NULL
applied	datetime	"applied" datetime NOT NULL
django_session		CREATE TABLE "django_session" ("session_key" varchar(40) NOT NULL PRIMARY KEY, "session_data" text NOT NULL, "expire_date" datetime NOT NULL)
session_key	varchar(40)	"session_key" varchar(40) NOT NULL
session_data	text	"session_data" text NOT NULL
expire_date	datetime	"expire_date" datetime NOT NULL
		CREATE TABLE sqlite_sequence(name,seq)

Name	Type	Schema
sqlite_sequence		
name		"name"
seq		"seq"
uNMS_alarm		
		CREATE TABLE "uNMS_alarm" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "clear_time" datetime NULL, "name" varchar(200) NOT NULL, "board" varchar(200) NOT NULL, "device_id" integer NOT NULL REFERENCES "uNMS_device" ("id") DEFERRABLE INITIALLY DEFERRED, "occur_time" datetime NULL)
id	integer	"id" integer NOT NULL
clear_time	datetime	"clear_time" datetime
name	varchar(200)	"name" varchar(200) NOT NULL
board	varchar(200)	"board" varchar(200) NOT NULL
device_id	integer	"device_id" integer NOT NULL
occur_time	datetime	"occur_time" datetime
uNMS_device		
		CREATE TABLE "uNMS_device" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "ipaddress" varchar(15) NULL, "type" varchar(20) NOT NULL, "model_id" integer NULL REFERENCES "uNMS_model" ("id") DEFERRABLE INITIALLY DEFERRED, "date_created" datetime NOT NULL, "city" varchar(200) NULL, "enterprise_id" varchar(200) NULL, "hostname" varchar(200) NULL, "online" bool NOT NULL, "region" varchar(200) NULL, "version" varchar(200) NULL)
id	integer	"id" integer NOT NULL
ipaddress	varchar(15)	"ipaddress" varchar(15)
type	varchar(20)	"type" varchar(20) NOT NULL
model_id	integer	"model_id" integer
date_created	datetime	"date_created" datetime NOT NULL
city	varchar(200)	"city" varchar(200)
enterprise_id	varchar(200)	"enterprise_id" varchar(200)
hostname	varchar(200)	"hostname" varchar(200)
online	bool	"online" bool NOT NULL
region	varchar(200)	"region" varchar(200)
version	varchar(200)	"version" varchar(200)
uNMS_interfaceoutboundoctacts		
		CREATE TABLE "uNMS_interfaceoutboundoctacts" ("interfaceindicatorcounter_ptr_id" integer NOT NULL PRIMARY KEY REFERENCES "uNMS_interfaceindicatorcounter" ("id") DEFERRABLE INITIALLY DEFERRED, "interface_id" integer NOT NULL REFERENCES "uNMS_interface" ("id") DEFERRABLE INITIALLY DEFERRED)
interfaceindicatorcounter_ptr_id	integer	"interfaceindicatorcounter_ptr_id" integer NOT NULL
interface_id	integer	"interface_id" integer NOT NULL
uNMS_interface		
		CREATE TABLE "uNMS_interface" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "device_ip" varchar(200) NULL, "ifType" varchar(200) NULL, "ifIndex" integer NOT NULL, "ifDescription" varchar(200) NULL, "ifPhysAddress" varchar(200) NULL, "ifAdminStatus" varchar(10) NULL, "ifOperStatus" bool NOT NULL, "ifSpeed"

Name	Type	Schema
		integer NULL, "ifMtu" integer NULL, "device_id" integer NOT NULL REFERENCES "uNMS_device" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
device_ip	varchar(200)	"device_ip" varchar(200)
ifType	varchar(200)	"ifType" varchar(200)
ifIndex	integer	"ifIndex" integer NOT NULL
ifDescription	varchar(200)	"ifDescription" varchar(200)
ifPhysAddress	varchar(200)	"ifPhysAddress" varchar(200)
ifAdminStatus	varchar(10)	"ifAdminStatus" varchar(10)
ifOperStatus	bool	"ifOperStatus" bool NOT NULL
ifSpeed	integer	"ifSpeed" integer
ifMtu	integer	"ifMtu" integer
device_id	integer	"device_id" integer NOT NULL
uNMS_interfaceinboundoctates		CREATE TABLE "uNMS_interfaceinboundoctates" ("interfaceindicatorcounter_ptr_id" integer NOT NULL PRIMARY KEY REFERENCES "uNMS_interfaceindicatorcounter" ("id") DEFERRABLE INITIALLY DEFERRED, "interface_id" integer NOT NULL REFERENCES "uNMS_interface" ("id") DEFERRABLE INITIALLY DEFERRED)
interfaceindicatorcounter_ptr_id	integer	"interfaceindicatorcounter_ptr_id" integer NOT NULL
interface_id	integer	"interface_id" integer NOT NULL
uNMS_interfaceindicatorcounter		CREATE TABLE "uNMS_interfaceindicatorcounter" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "time" datetime NOT NULL, "value" integer NULL)
id	integer	"id" integer NOT NULL
time	datetime	"time" datetime NOT NULL
value	integer	"value" integer
uNMS_interfaceperformance		CREATE TABLE "uNMS_interfaceperformance" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "utilization" bool NOT NULL, "inoctates" bool NOT NULL, "outoctates" bool NOT NULL, "interface_id" integer NOT NULL REFERENCES "uNMS_interface" ("id") DEFERRABLE INITIALLY DEFERRED, "device_id" integer NOT NULL REFERENCES "uNMS_device" ("id") DEFERRABLE INITIALLY DEFERRED, CONSTRAINT "unique_interface" UNIQUE ("interface_id", "device_id"))
id	integer	"id" integer NOT NULL
utilization	bool	"utilization" bool NOT NULL
inoctates	bool	"inoctates" bool NOT NULL
outoctates	bool	"outoctates" bool NOT NULL
interface_id	integer	"interface_id" integer NOT NULL
device_id	integer	"device_id" integer NOT NULL
uNMS_interfacetrap		CREATE TABLE "uNMS_interfacetrap" ("occur_time" datetime NULL, "clear_time" datetime NULL, "interface_id" integer NOT NULL PRIMARY KEY REFERENCES "uNMS_interface" ("id") DEFERRABLE INITIALLY DEFERRED, "linkStatus" bool NOT NULL, "name" varchar(200) NOT NULL)
occur_time	datetime	"occur_time" datetime

Name	Type	Schema
clear_time	datetime	"clear_time" datetime
interface_id	integer	"interface_id" integer NOT NULL
linkStatus	bool	"linkStatus" bool NOT NULL
name	varchar(200)	"name" varchar(200) NOT NULL
uNMS_interfaceutilization		CREATE TABLE "uNMS_interfaceutilization" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "value" real NULL, "time" datetime NOT NULL, "interface_id" integer NOT NULL REFERENCES "uNMS_interface" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
value	real	"value" real
time	datetime	"time" datetime NOT NULL
interface_id	integer	"interface_id" integer NOT NULL
uNMS_model		CREATE TABLE "uNMS_model" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(200) NOT NULL, "model_index" integer NULL, "vendor_id" integer NOT NULL REFERENCES "uNMS_vendor" ("id") DEFERRABLE INITIALLY DEFERRED)
id	integer	"id" integer NOT NULL
name	varchar(200)	"name" varchar(200) NOT NULL
model_index	integer	"model_index" integer
vendor_id	integer	"vendor_id" integer NOT NULL
uNMS_snmpdetails		CREATE TABLE "uNMS_snmpdetails" ("rcommunity" varchar(200) NOT NULL, "timeout" integer NOT NULL, "retries" integer NOT NULL, "security_level" varchar(3) NOT NULL, "device_id" integer NOT NULL PRIMARY KEY REFERENCES "uNMS_device" ("id") DEFERRABLE INITIALLY DEFERRED, "wcommunity" varchar(200) NOT NULL, "version" integer NOT NULL)
rcommunity	varchar(200)	"rcommunity" varchar(200) NOT NULL
timeout	integer	"timeout" integer NOT NULL
retries	integer	"retries" integer NOT NULL
security_level	varchar(3)	"security_level" varchar(3) NOT NULL
device_id	integer	"device_id" integer NOT NULL
wcommunity	varchar(200)	"wcommunity" varchar(200) NOT NULL
version	integer	"version" integer NOT NULL
uNMS_vendor		CREATE TABLE "uNMS_vendor" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(200) NOT NULL, "vendor_index" integer NOT NULL)
id	integer	"id" integer NOT NULL
name	varchar(200)	"name" varchar(200) NOT NULL
vendor_index	integer	"vendor_index" integer NOT NULL

Indices (25)

Name	Type	Schema
auth_group_permissions_group_id_b120cbf9		CREATE INDEX "auth_group_permissions_group_id_b120cbf9" ON "auth_group_permissions" ("group_id")
group_id		"group_id"
		CREATE UNIQUE INDEX

name	type	Schema
tent_type_id_c4bce8eb		"django_admin_log" ("content_type_id")
content_type_id		"content_type_id"
django_admin_log_user_id_c564eba6		CREATE INDEX "django_admin_log_user_id_c564eba6" ON "django_admin_log" ("user_id")
user_id		"user_id"
django_content_type_app_label_model_76bd3d3b_uniq		CREATE UNIQUE INDEX "django_content_type_app_label_model_76bd3d3b_uniq" ON "django_content_type" ("app_label", "model")
app_label		"app_label"
model		"model"
django_session_expire_date_a5c62663		CREATE INDEX "django_session_expire_date_a5c62663" ON "django_session" ("expire_date")
expire_date		"expire_date"
uNMS_alarm_device_id_d589da7d		CREATE INDEX "uNMS_alarm_device_id_d589da7d" ON "uNMS_alarm" ("device_id")
device_id		"device_id"
uNMS_device_model_id_2eacf09b		CREATE INDEX "uNMS_device_model_id_2eacf09b" ON "uNMS_device" ("model_id")
model_id		"model_id"
uNMS_interfaceoutbounddoctates_interface_id_c1b5a875		CREATE INDEX "uNMS_interfaceoutbounddoctates_interface_id_c1b5a875" ON "uNMS_interfaceoutbounddoctates" ("interface_id")
interface_id		"interface_id"
uNMS_intf_device__71ed28_idx		CREATE INDEX "uNMS_intf_device__71ed28_idx" ON "uNMS_interface" ("device_id", "ifIndex")
device_id		"device_id"
ifIndex		"ifIndex"
uNMS_interface_device_id_5805df12		CREATE INDEX "uNMS_interface_device_id_5805df12" ON "uNMS_interface" ("device_id")
device_id		"device_id"
uNMS_interfaceinbounddoctates_interface_id_1a2c2734		CREATE INDEX "uNMS_interfaceinbounddoctates_interface_id_1a2c2734" ON "uNMS_interfaceinbounddoctates" ("interface_id")
interface_id		"interface_id"
uNMS_interfaceperformance_device_id_c7359c45		CREATE INDEX "uNMS_interfaceperformance_device_id_c7359c45" ON "uNMS_interfaceperformance" ("device_id")
device_id		"device_id"
uNMS_interfaceperformance_interface_id_133bd50d		CREATE INDEX "uNMS_interfaceperformance_interface_id_133bd50d" ON "uNMS_interfaceperformance" ("interface_id")
interface_id		"interface_id"

Scheduled Tasks

```
from __future__ import unicode_literals, absolute_import
from NMS.celery import app
from snmp_cmds import Session, exceptions
from .library.snmp_manager import Manager
from .unmslibrary import DeviceObject, SNMPTrap

@app.task(autoretry_for=(exceptions.SNMPTIMEOUT,),max_retries=4, default_retry_delay=5)
def listInterfaces(host):

    device = Manager(host)
    # list = device.device_session.get_table(oid='IF-MIB::ifTable')
    list = device.list_interfaces()

    return list

@app.task()
def check_all_device_heartbeat():

    device_manager = DeviceObject() # create a device object manager instance
    devices = device_manager.all() # query all devices configured in the system

    for device in devices:
        device_heartbeat.delay(device.ipaddress) # call heartbeat task per device

@app.task(autoretry_for=(exceptions.SNMPTIMEOUT,),max_retries=5, default_retry_delay=6)
def device_heartbeat(host):

    device = Manager(host)
    try :
        sysname = device.systemname()

    except exceptions.SNMPInvalidAddress:
        return {host:'INVALID'}

    except exceptions.SNMPError:
        device.device_obj.setOffline()
        return {host:'ERROR'}

    except exceptions.SNMPTIMEOUT:
        device.device_obj.setOffline()
        return {host: 'OFFLINE'}

    device.device_obj.setOnline()
    return {host:'ONLINE'}

@app.task()
def savetrap(data):
    """
    Task to save a received trap. This task invokes methods from the unmslibrary for processing and
    saving the trap
    recieved. The trap before being saved needs to be parsed to check if device is configured in
    system, enterprise and
    OID.
    :param data:
    :return:
    """
    thetrap = SNMPTrap(data) # initialis SNMP trap object
    thetrap.savetrap() # save the trap into database
    pass

@app.task()
def device_update_int(host):
    """
    Task to update device properties in the database
    :param host:
    :return: None
    """
    device = Manager(host)
```

```

device.update_systemname_db() # update the system name
device.update_device_enterprise_id() # update the enterprise value
device.update_interface_db() # update the interface DB

@app.task()
def update_all_device_int():

    device_manager = DeviceObject() # create a device object manager instance
    devices = device_manager.all() # query all devices configured in the system

    for device in devices:
        device_update_int.delay(device.ipaddress) # call interface update task per device

@app.task()
def update_all_device_performance():

    device_manager = DeviceObject() # create a device object manager instance
    devices = device_manager.all() # query all devices configured in the system

    for device in devices:
        collect_device_performance.delay(device.ipaddress) # call the collect performance

@app.task()
def collect_device_performance(host):
    device = Manager(host) # create manager instance for the device

    device.collect_interface_performance() # collect performance

```

Configured URLs for the project

```
from django.conf.urls import include, url
from django.urls import path

from . import views

from uNMS.views import *

app_name = "uNMS"
urlpatterns=[
    path('', DeviceList.as_view(),name='landing_page'),
    path('vendoradd/', CreateVendor.as_view(),name='create_vendor'),
    path('vendorlist/', ListVendor.as_view(),name='list_vendors'),
    path('vendordetail/<str:pk>', DetailVendor.as_view(),name='detail_vendor'),
    path('vendorupdate/<str:pk>', UpdateVendor.as_view(),name='update_vendor'),
    path('vendordel/<str:pk>', DeleteVendor.as_view(),name='delete_vendor'),
    path('modeladd/', CreateModel.as_view(),name='create_model'),
    path('deviceadd/', CreateDevice.as_view(),name='create_device'),
    path('deviceupdate/<str:pk>', UpdateDevice.as_view(),name='update_device'),
    path('deviceupdate/<str:pk>/delete', DeleteDevice.as_view(),name='delete_device'),
    path('devicelist/', DeviceList.as_view(),name='list_device'),
    path('devicelist/offline/', OfflineDeviceList.as_view(),name='list_offline_device'),
    path('devicelist/online/', OnlineDeviceList.as_view(),name='list_online_device'),
    path('snmpadd/', DefineDeviceSNMP.as_view(),name='create_snmpprofile'),
    path('snmpupdate/<str:pk>', UpdateDeviceSNMP.as_view(),name='update_snmpprofile'),
    path('snmpdetail/<str:pk>', DeviceSNMP.as_view(),name='detail_snmpprofile'),
    path('snmpdetail/', ListSNMPProfiles.as_view(),name='list_snmpprofile'),
    path('deviceinterfaces/<str:pk>', DeviceInterfaces.as_view(),name='device_interface_list'),
    path('devicemonitoring/<str:pk>', DeviceMonitoring.as_view(),name='device_monitoring'),
    path('ifmon/<str:pk>', InterfaceMonitoring.as_view(),name='interface_monitoring'),
    path('ifmonconf/<str:pk>',
InterfaceMonitorConfigurationView.as_view(),name='interface_monitoring_config'),
    path('ifmondel/<str:pk>', DeleteIFMonitoring.as_view(),name='interface_monitoring_delete'),
    path('updateifmon/<str:pk>',
UpdateIFMonitoring.as_view(),name='update_interface_monitoring_config'),
    path('devmon/<str:pk>', DeviceMonitoringView.as_view(),name='device_monitoring_view'),
    path('interfacealarms', InterfaceTraps.as_view(),name='interface_trap_list'),
    path('dashboard', DashboardView.as_view(),name='dashboard'),
    path('ciscodashboard', CiscoDashboardView.as_view(),name='dashboard_cisco'),
    path('ztedashboard', ZTEDashboardView.as_view(),name='dashboard_zte'),
    path('hwdashboard', HWDashboardView.as_view(),name='dashboard_huawei'),
]
```

Snmpttrapd.py file

```
#!/usr/bin/env python3
from __future__ import absolute_import, unicode_literals
import os
from celery import Celery
import sys
import fnmatch
import datetime

app = Celery('snmpttrapd', broker='redis://localhost:6379/0')

def varbind_parser():
    lines = sys.stdin.readlines()
    linecount = len(lines)
    i = 2 # skip the host and ip entries
    varbinds = dict()
    while i <= linecount-1:
        line = (lines[i].split())
        varbinds[line[0]] = line[1]
        i=i+1

    time = datetime.datetime.now() # get current time
    varbinds['time'] = time.__str__() # add time to varbinds
    varbinds['IPADDRESS'] = lines[1] # assign ip address
    return varbinds

def retrieve_key(dictionary,key):
    keys = list(dictionary)
    key = key + '*'
    for k in keys:
        if fnmatch.fnmatch(k,key):
            return k

def listint(host):
    #function to list interfaces

    app.send_task('uNMS.tasks.listInterfaces', (host,))

if __name__ == "__main__":
    data = varbind_parser() # receive varbinds from snmpttrapd daemon
    app.send_task('uNMS.tasks.savetraps',(data,)) # create a task to save the trap received
```