BACK-END APPLICATION FOR MONITORING MESH NETWORK

By

Ajayi Olabode Oluwaseun

A thesis submitted in partial fulfillment of
the requirements for the degree of

Baccalaureous Scientiae (Honours)

University of the Western Cape

Supervisors: Prof. Bill Tucker

Co-Supervisor: Mr. Michael Norman

Assisted by: Mr. Carlos Rey-Moreno and Mr. A. Kruger

Date: 06 November 2013

University of the Western Cape
Abstract

BACK-END APPLICATION FOR MONITORING MESH NETWORK

By

Ajayi Olabode Oluwaseun

Supervisors: Prof. W.D Tucker
Co-Supervisor: Mr. M. Norman
Department of Computer Science

Wireless mesh networks are complex to monitor and institutions such as University of the Western Cape is finding some way to solve the complexity problems facing wireless mesh network activity. Managing and monitoring remote or local internet connection on University campus and rural area (e.g. Eastern Cape, South Africa) involves high cost of maintenance and the cost of installing physical equipment. Thus, mesh network have the advantage of easy-to-deploy and fault-tolerant. Organization believes that wireless technologies are cost-effective and scalable. Students cell phone or end user devices can connect anytime anywhere on the network. To manage and monitor mesh network activity on University campus and rural area we propose a backend and frontend application system. The frontend application of this project will utilize a wireless mesh network topology visualization tools to monitor the mesh network activities on the network. The back-end application of this project will obtain network configuration information from different Mesh Potatoes devices and the obtained values or data will be stored in a file, and its size will be managed. The backend application will collect information at different moments in time according to the granularity defined on each mesh node. Particularly, the output values (data stored) obtained on each mesh node will be parse to a centralized database server when the traffic on the network is expected to be low (at night) and the log files will be aged when acknowledged by the server. Therefore, the propose system will demonstrate a practical complete system that offers solutions to the complexity, and provide solutions to some key areas of the network problems while carry out network monitoring and maintenance

TABLE OF CONTENTS.

LIST OF FIGURES

Number *Pages*

LIST OF TABLES

ACKNOWLEDGEMENTS

I take this opportunity to express my thoughtful gratitude and profound regards to my supervisor Professor Bill Tucker, and my co-supervisor Mr. Michael Norman, Mr. Carlos Rey-Moreno and Mr. Andre Kruger in the Computer Science department for their great guidance and constant encouragement through the course of this first part of my project thesis delivery. Your encouragement towards this project and where I am, where I will be, and where I am going shall go a long way in the journey of my life. In addition, I take this opportunity to express my deep sense of gratitude to Thrip, Centre of Excellence (CoE) for sponsoring my honours project. My most respected brother and reliance friend, and guardian Dr. T.E Oni, I am very happy to have come across you and your family. My encounter with you has brought in success. It has, and will always give me hope to live up to my dreams and achievement. My prayer to you and your family is that God will raise you beyond your doubt and fear, and you will be celebrate for as long you are alive. I will never forget that moment you clothed me, sheltered me, and taking care of me as a son and brother. You are the best. God bless you and your house-hold.

Lastly, I thank Almighty, my parents, brother, sisters and friends for their constant encouragement without which this project would not be possible.

Yours Sincerely,

[illegible]

Ajayi Olabode Oluwaseun

GLOSSARY

API	Application Programming Interface
ETL	Extraction Transformation and Loading
SS	Source System
TS	Target System
B.A.T.M.A.N-ADV	Better Approach To Mobile Adhoc Networking Advance
DBI	Database Interface
DBD	Database Driver
ODBC	Open Database Connectivity
MP	Mesh Potatoes
Wi-Fi	Wireless
WLAN	Wireless Local Area Network
OS	Operating System
SCEN	Small Campus Enterprise Network
CLI	Command Lines Interface
LAMP	Linux Appache MySQL PHP

Chapter 1

PROJECT PROLOGUE

Introduction

A mesh network is a wireless local area network (WLAN) where each node is connected to others nodes. Wireless mesh networks are configured to allow wireless connections to be routed around fragmented paths with each signal hopping from node (source) to other node until it reaches their destination. Wireless mesh network is self-healing and self-configuring. Self-healing is the ability of the wireless mesh node or Mesh Potatoes (MP) to be perceive as not operating correctly and, without human involvement, it then further make the necessary changes to restore itself to normal operation. When such nodes in the network become broken, self-healing mechanisms aim at reducing the impacts from the failure, for example by adjusting parameters in mesh mode using some configuration commands so that other nodes can support the users that were supported by the failing node. Likewise, self-configuration strives towards the plug-and-play paradigm in the way that new mesh typology setup shall automatically be configured and integrated into the network. However, wireless mesh networks are difficult to monitor and its network activity (such as performance) is difficult to manage and maintain, also protocols and services are becoming more complex to manage.

Hence, we propose a system to help addresses and proffer solutions for solving the complexity facing mesh network.Most importantly, wireless mesh network operate between layers 1 and layer 3. They are packet-switched networks with a static backbone [1]. More so, wireless mesh network (i.e. Wireless-Fidelity (Wi-Fi)) nodes which comprises of Access point (AP), and clients access (CA). Each mesh node supports wireless connections that are cost-effective, and a convenient way to setup a mesh mode network for department (e.g UWC Computer Science Department), and in rural area (e.g. Mankosi in Eastern Cape, South Africa) that don’t have internet connections.

In addition, each node will operate not merely as a host but also as routing devices (Mesh Potatoes) that will serve as either a gateway or remote gateway to assist in Wi-Fi connection. Using these Mesh-Potatoes as a router (Gateway) on the network will provide a secure data transmission and will help to forward packets to other nodes that may not be within uninterrupted wireless transmission range of their destinations [2].

The goal of this project will be to create a backend application that will enable the management of network data, the configuration of a remote gateway on mesh network, and the rate used with each neighbor on active link quality. Moreover, for this project to complete, we also propose a frontend application that will use visualization tools to display nodes information as well as icons on a topographical map and some indication of link quality between nodes.

Additionally, the application is to overlay the entire network display on a geographical map, and provide node information when the corresponding node icon is clicked on. These are important features on network visualization, and are something which SCUBA lacks [3]. This project will enable the integration of wireless mesh networks (Wi-Fi) activity with database management system to help feed the frontend visualization displacement [3,4].

Motivation

Like other institutions e.g. the University of the Western Cape has seen the usage of wireless technology blowup over the past few years. It was not long ago that some universities discover the use of wireless technology. Wireless mesh networks requires proper network database management system for its data transfer or communication. The need for a frontend and backend monitoring application system for wireless mesh network is to save the network manager from manually storing and analyzing network accounting information data. Since monitoring mesh network is complex, the project proposes to develop a system that will help the network manager to monitor and to help solve the complexity facing mesh activity. For instance, the propose system will help the network manager to minimize the space occupied by the log file. After that, by ageing the log file it from the network device (Mesh Potatoes) when outputting data information for decision making process.

More so, the propose system will help frequently checking the values or mechanisms to compress the data explored before they are send to the centralized database server. The system will determine the flexibility of packet exchange between each node on the network. A poor link quality between two routers could indicate a problem with one or both of the routers, and hence a problem in the network can be identified. Hence, the representation of link quality in a database server can aid in fault management (because problems in the network can be identified), performance management (network performance can be improved if poor-quality links are discovered in routing table), and accounting management (a poor-quality link could also indicate an extremely high amount of network traffic on that link).Thus, because of the growing number of data on mesh devices, this project will develop an application system, particularly a backend application for monitoring mesh network activity.

More so, this project proposes to use an agile methodology approach. The reason for this approach is to manage changes and reduces risk management at any time anywhere. Therefore, this project finally proposes to help the network manager to solve and evaluate the network metrics, and determine the quality of the links on the network.

Planning a Mesh Network

For effective communication and information sharing institutions (e.g. UWC) are increasingly relying on computer networks and communication tools, such as computers software. This project will set up a mesh mode network that will enable the network manager to access remote networks, the databases server, and other applications within the same network as well as other private or public networks. However, implementing this set up is a complex task. This project will need to consider the goals of the institution e.g. UWC or the rural area in Makosi Eastern Cape while making a decision on which accounting information is needed.

This project will also consider the network infrastructure which consists of the physical and logical components that are required to meet the networking needs of an organization. The physical components of the mesh network infrastructure are computers, servers, routers (Mesh Potatoes), switches and network cables. The logical components are the software that will be used to enable the flow of data transfer or communication across the mesh network.

Monitoring of Mesh Network

This project proposes some network parameters, that is, how to use and setup a mesh network with b.a.t.m.a.n-adv. The idea is to collect and store these network values and use it to monitor the mesh network activity. For instance, “batctl” holds the commands “ping, traceroute, tcpdump” that provides a suitable way to configure the batman-adv kernel module as well as showing debug information (such as originator tables, translation tables and the debug log) [4]. This project will further configure interfaces, check the quality of links and performance, nodes connectivity, and rate info. However, this project will limit its discussion on some batman-adv commands only as the full topic is beyond the scope of this project.

Chapter 2

REQUIREMENTS DOCUMENT

Introduction

This chapter discusses and presents the user requirement for the proposed system. It explains what the user view of the problem is, and briefly describe the problem domain. The chapter further explains what is expected, and what constraints are applicable. More so, this chapter was compiled as a result of several literature reviews and information gathered from the University of Western Cape library repository.

User's (Network Manager) View of the Problem

Monitoring mesh network activities can sometime be complex. This ranges from obtaining and storing network values, different instances of the value, links quality, and performance. Some network parameters are kept and cause huge amounts of data on the mesh network. Currently, institution such as UWC network manager is using a bash scripts on the mesh router by executing it to collect data and statistics information on the network and store them in a text file. This can be referring to as log file. The data collected are huge and the network manager only require less of the data for managing and monitoring the mesh network activity as well as for troubleshooting and configuring the remote network nodes.

The typical data required by the network manager would be the values outputting from Mesh Potatoes when debugging using the following batman-adv commands: *athstats* and *ifconfig* to do some aggregation, and *batctl o, wlanconfig, ath0 list, and rate info parameters* for doing the dynamic routing that depict link quality and performance. The network manager will want to inject traffic, copy the data and view the information to have a continual awareness and to avoid traffic jam or congestion.

In addition, the network manager will want a centralize database server to manage the network values that are collected in order to feed the frontend of the proposed system. The relevant data that are being stored in the centralized database server will be presented to the frontend application to allow the network manager to monitor and visualize the network activity within the internal nodes of the network.

Brief Description of the Problem Domain

A user such as a network manager needs a network monitoring system to help manage and monitor mesh network activity. The network manager will execute different network protocol commands to collect data and store them into a centralize database. The used of a configuration files will help the network manager to populate the centralize database server with the most relevant data and to keep track of different parameters used during network debugging. More so, the network manager is looking for a way to curtail the time expending on collecting data from each mesh nodes, and to increase the effectiveness of the data transfer between a node and the centralized database servers.

What Is Expected From the Software Solution

The software is expected to establish a remote access network connection between the local organization (e.g. UWC) and the remote site (e.g. rural area) where network activity are taken place and can be monitor. The network values or executed parameters for connection is expected to utilize low resource memory space of Mesh Potatoes devices, be secure and reliable, and increase the effectiveness of the data transfer between the routing protocols (batman-adv) and the centralized database servers. More so, the network manager is expecting the software to manage and monitor mesh network activity in an effective way that is cheaper and affordable by the institution and rural community. The software should investigate the batman-adv routing protocol that will offer an essential diverse approach to route network traffic of each nodes to any destination in the network.

Similarly, the software is expected to help the network manager to save time expending on data collection. That is, the more time the network manager requested output from the mesh network activity the better the quality of the information he will get or have to make decision. Therefore, the software is expected to save the device storage space (of the mesh potatoes) to a minimum level when analyzing and populating the centralize database server with the most relevant data information.

Constraints

The software is not expected to override existing system functionality. It is important that this is taken into consideration when planning and designing the software solution. The use of the software should not disrupt the network activity, and connection between each node on the network when sending or receiving packets from source and target system. More so, the software is not expected to consume device space and network resources.

System Users (Network Manager)

The network managers or system administrators are the people who will be using the system to perform administrative tasks. They will be able to efficiently administer the network with a minimum of personnel and stress. Likewise, they will gain a broad, cost-effective view of what is required to setup an efficient software application for monitoring network activity. In addition, the software developers are fundamental in this project as they are the primary programmer of this particular software.

Therefore, this software for monitoring mesh network activity, it will help and target the time network manager expends collecting and aggregating data by reducing it to a minimal level and also reduces the manually made mistakes when computing the most important data for decision-making process.

Chapter 3

REQUIREMENTS ANALYSIS DOCUMENT (RAD)

Introduction

This chapter discusses the requirements analysis of the system using the Village Telco device (the Mesh Potato routers) as a key factor for requirement analysis. The chapter further discusses the hardware, software and high level requirements needed to implement the user requirements of the problem domain.

Designer's interpretation of the user's requirements

The following diagram figure 1 will depict the network design requirement (network infrastructure diagram) for the proposed backend application system. Since a user (Network manager) wants systems that will help him carry out daily network monitoring a mesh mode infrastructure will be set-up to address how the user's requirement will be implemented later at implementation stage of this project.

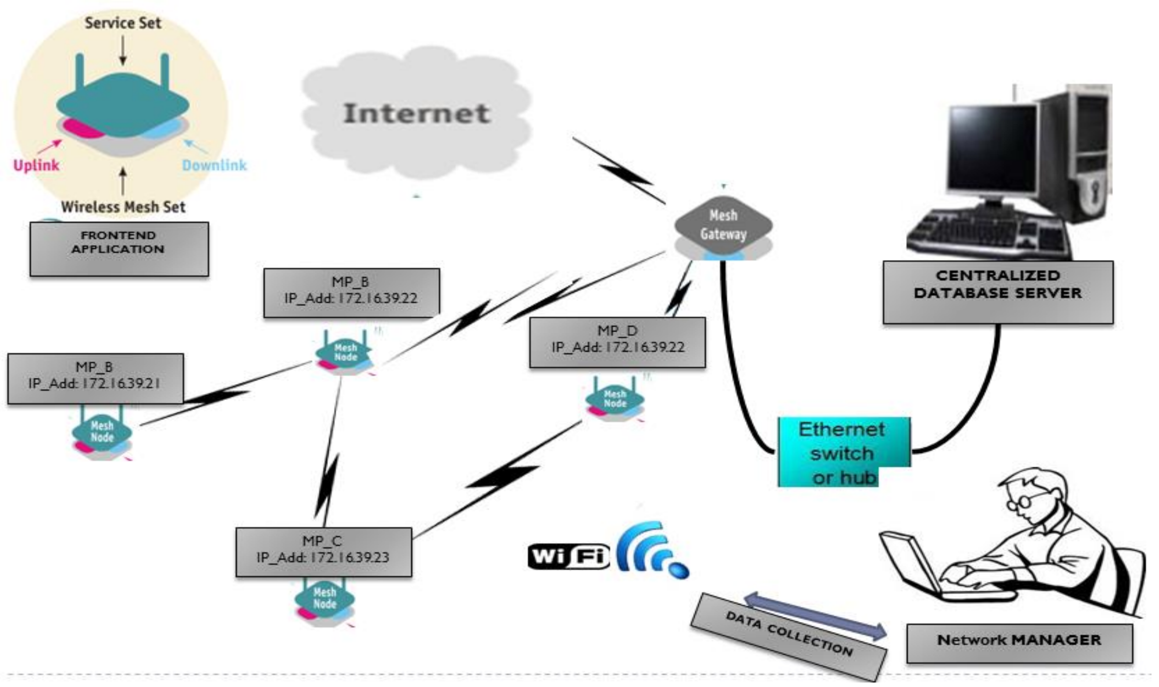


Figure 1: Diagram showing an illustration of designer's

Interpretation of problem domain

The above figure shows a backend network infrastructure diagram, to the right of the design requirement the network manager was trying to configure a router (Mesh Potatoes) on the wireless mesh network using Linux as the operating system. On the left of the diagram, a wireless mesh network was setup to allow wireless local area network (WLAN) or Wi-Fi connectivity so that the network manager who wants to configure, monitor or query all nodes on the network for their connectivity information can configure the router locally or remotely using the remote access gateway.

Each node MAC address will be marked with their hostname to enable easy node identification and for easy lookup of the IP address (e.g. 172.16.*.*). Moreover, the network manager will want to examine and monitor the quality of the link, node connection and signal strength. This setup can be seen to be a complex network setup however this project proposes a solution to solve the network complexity. The network parameters set up will be break down and will be analyzed. After breaking down the problem, the information from the network will be computed into database management system to assist the frontend application system.

This suggested solution will best suit the operation and running process of the wireless mesh network. That is, organization will be able to increase the number of routers as per the requirements and increased the network traffic. Each node will learn routes using a very stigmeric approach [1].

System Architecture

The system architecture for this project (e.g. backend application for monitoring mesh network activity) will comprised of two main components namely; the hardware configuration and the software configuration package. These two components will be outlined and then broken down into more defined sub-components at a later stage of the software development life cycle.

This project will employ the concept of data mining using Extract, Transform, and Load (ETL) for its database management system. Each sub-component will be responsible for the remote or local configuration for mesh networks. The hardware component will further discuss the hardware of the mesh devices (Mesh Potatoes) and the software necessary for the data collection for storing the networks values into the centralized database server. In the same way, the software component will be used to gather the required necessary network accounting information that the network manager wants to see on the frontend application. This backend application is expected to help and make available information needed to supply the front end monitoring system for easy access, nodes visualization, and monitoring. However, to gather the required network information, it is important to ensure that the mesh devices, computers system, and switches are part of the network and have the right configuration setup.

Therefore, nodes on the network will communicate with each other nodes in the network. More so, a Perl program will be used in this project to help query and analyze the entire mesh node on the network for relevant data information. Such information will include network of the next neighbors, link quality on the mesh network and connection strength to neighbors, network traffic, and rate info for maximum and minimum throughput of the nodes on the network. The following figure 2 shows a sample of system architecture to be implemented for this project:

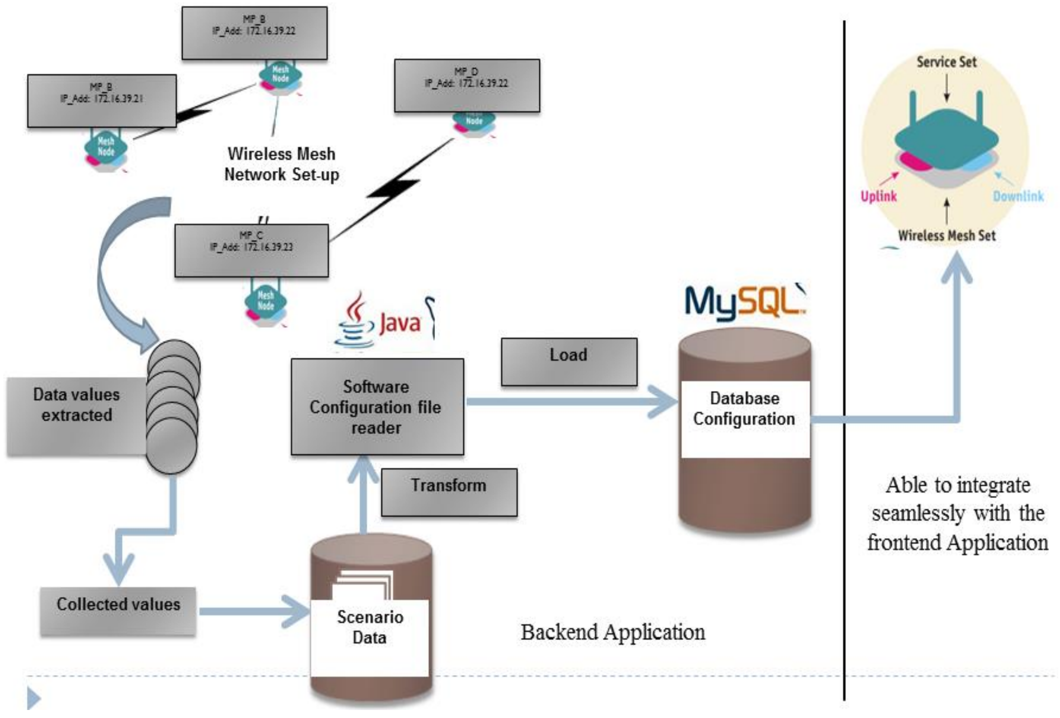


Figure 2: Illustrating the System Architecture for Requirements Analysis

Hardware Configuration Component

The hardware configuration component will include the mesh potatoes device, switches, network cables, and computer system. This part of the project will be implemented incrementally to tackle the network manager view of the problem. The fundamental sub-components in this project include mesh potatoes and the backend hardware's.

Mesh Potatoes (MP)

The mesh potato which is a wireless router will be connected to the switch, and the computer system will be part of the network to form an ad hoc mesh network. This ad hoc mesh network will enable the execution and evaluation of the back end application for monitoring the network activity. The diagram illustrated in the table 1 shows each network node and their main contribution to the ad hoc wireless mesh network.

Furthermore, the mesh potatoes devices utilize an Open System Interconnection (OSI) Layer 2 protocol, even layer 1 of the OSI protocol and will basically act as one main router (Gateway), transparently connecting all the attached devices together. Mesh Potatoes have the Village Bus module installed, which act as an interface for all the data stored on them, such as their accounting information and connection to neighbors. The hardware specifications of the Mesh Potatoes routers are given below in the table 1:

Hardware	**Details**
Processor	MIPS 4k 180 MHz
Memory	16Mb Ram, 8Mb flash EEPROM
Wireless LAN, WiFi	IEEE 802.11b/g 2.4 to 2.462 GHz frequency band Omni directional Antenna
Wireless Configuration	Ad hoc mode
Firmware	Linux kernel 2.26.3 Small Campus Enterprise Network software (SCEN) or Customized OpenWRT, B.A.T.M.A.N Advance routing potatoes

Table 1: Hardware Specifications of the Mesh Potatoes Routers

The Mesh Potatoes hardware configuration and software management in the table above will be accessible via browser or Linux terminal sessions with an access to the core Linux operating system kernel, OpenWRT or the Small Campus Enterprise Network software (SCEN). Mesh potatoes use batman-Advance (as seen above in the table) for its mesh network routing protocol. A number of these devices will be used in this project to provide data for the network activity. Each MP device provides Wi-Fi Access Point (AP) for connection, and allows collection of log file automatically using a scheduler tasks e.g. a scripting program (crons) which is responsible for optimizing the collection various logs, and also for analyzing the mesh activities. An Ethernet cable will be used for communicating and to connect the Mesh Potatoes to a source system. This will allows the network manager to setup the wireless local area network (WLAN) and Wi-Fi gateway.

In addition, a personal computer (PC) and a network cable will be connect to a network switch using the Ethernet port of the Mesh Potatoes node to gain access to Wi-Fi which will enable the network manager to connect anytime anywhere. One key factor that is essential to take note in this project is that once a Mesh Potatoes device is connected via Ethernet cable to a LAN router (MP), then devices on the LAN will gain access to local or Internet connection as well as gain access to the LAN resources to help the network manager to connect to any of the ports on the network.

Backend Database Servers (MP_DB Server)

The backend database server is a central repository component of this backend application system and it will help with storing and retrieving capability which will allows the network manager to insert, update, and query nodes that are on the ad hoc mesh network [3]. A computer system will be setup to cater for the operation and running of the mesh network. The backend database server will have the following specifications: IBM Sun Solaris Ultra 20 Workstation, AMD Opteron 64bit, 1800 +CPU, 1GB RAM and 80GB SATA2 hard drive. This database server will be able to manage a very large data communication on the mesh network and also requires constant network monitoring and retrieval. However, research shows that IP networks are difficult to manage (a side effect of their decentralized nature); protocols and services are becoming more complex [5].

Furthermore, through this backend database server its primary goals are to minimize storage space or memory space of the device by ageing the log file on the MP and hence, this will be done during the local or remote network configuration set-up. Analysis of network traffic and the performance log file during monitoring and debugging will also be kept at a reduced space on the Mesh Potatoes. In addition, this project has foreseen the need for a structure query environment and adaptability which is most evident to the network manager.

One significant problem with mesh network is the high rate of data generating per day hence, if the analysis of data could be done on-the-fly, this project will offer capabilities that current system such as SPUD [6] cannot provide. The backend application server would be lightweight stream query processing systems instead of the traditional database type (one that cannot handle large data). This system will be at least be as fast as a hand-written system when querying the database to present at the frontend application which will allows the user (network manager) to by-pass the query system as needed.

Software Configuration Package Component

This project will implement a software application for downloading and collecting the mesh network data. This collected data will be analyzed and stored into the backend database server for easily accessibility for the frontend application. How this will be achieved will include the following sub-components lists namely: Small Campus Enterprise Network (SCEN) or OpenWRT, Application File Extractor module, Better Approach to Mobile Ad hoc Networks (B.A.T.M.A.N-ADV), and MySQL Database Application Software. Brief description of the sub-component lists is as follows:

Mesh Potatoes Small Campus Enterprise Network (SCEN) or Openwrt Firmware

This project will use the Small Campus Enterprise Network firmware that is designed to allow a connection of Mesh Potato devices to provide a data and Internet network for a small campus. The intended use will typically be for a small/medium size organization which needs to set up mesh network for a geographical area. This will allow the network infrastructure to be wirelessly connected without the use of conventional LAN cabling.

The meshed MP devices will utilize an OSI Layer 2 protocol and act as one large switch that will connect all other network devices together. In this project, each MP device will provide an Ethernet cable connection, and a Wi-Fi Access Point. PCs and other network devices will be connected to the Ethernet port of a Mesh Potatoes, or they will be connected wirelessly to the Wi-Fi Access Point of each Mesh Potatoes.

Application File Extractor Module

An application file extractor module will be programmed using Perl and bash scripting language. This will help load data into the database server. It will also extract the relevant data from the Mesh Potatoes devices (configuration files) to minimize and save the Mesh Potatoes memory space. The file extractor will help to collect relevant data logs and usage information from the mesh network activity and load them in the back end database to help the network manager for making decision based on the quality of link (QoS) getting from the ad hoc mesh network.

Better Approach to Mobile Ad hoc Networks (b.a.t.m.a.n-adv)

In this project, the B.A.T.M.A.N-ADV routing protocol is a form of a Linux kernel module operating system and operate at ISO/OSI layer 2 level [2, 6] protocol. This routing protocol will proactively maintain information about the existence of all nodes on the network that will be accessible via single-hop or multi-hop communication links. The main purpose of routing protocol in this project is to help determine for each destination on the mesh one single-hop neighbor or multi-hop links. These links (neighbor IP address) will be utilized as a best gateway to communicate with the destination node.

However, how the batman-adv routing protocol will be achieved in this project will be that all nodes will periodically broadcast packets that are known as originator messages to its neighbors. The originator messages will consist of an originator address, sending node address and a unique sequence number. Each neighbor changes the sending address to its own address and re-broadcasts the message. While at the receiving end of the mesh node, the originator does a bidirectional link check to verify that the detected link can be used in both directions. Batman-adv will optimize data flow through the mesh potatoes when injecting traffic or correcting forward errors.

MySQL Database Application Software

MySQL will help with the project implementation. These include running MySQL database on the Linux operating system to serves as the main server to the source system and then supporting the file extractor modules. This means the file extractor module will directly be connected to the MySQL database to help in the frontend implementation. All relevant data values will be collected and loaded into MySQL database to facilitate the frontend application system for monitoring the mesh network activity.

High-Level Design of the Solution

The network manager will interact with the backend system to troubleshoot, configure and manage the Mesh Potatoes (MP) router. The network manager will further collect the necessary values to store in the central database. This part of this project will show the internal overview of the routing protocols for mesh-node with BATMAN-ADV configuration snippet on the Small Campus Enterprise Network. The high-level configuration design will include the key data collection design, key routing protocol functions and the expected behaviors of wireless mesh network.

Data Collection Design

This sub-part of this project will include collecting the values that are relevant to the managing and monitoring of the mesh network activity. This will help the network manager to facilitate easy network communications and the link quality of each network node. However, this data collection design will be an open arena for possibilities to make changes with regard to the distribution of data collected from each MP on the network for populating the database. Whenever possible and appropriate, the database designs structures will consider and allow direct access of the target system to store data collected from either local or remote configurations.

Routing Protocol Functions

The following figure 3 illustrates routing scenario using an asymmetric routing when setting up b.a.t.m.a.n-adv routing concept [2]. This project will use this concept to design a wireless mesh network of each node such as; node A, B and C are connected via asymmetric links then, every node will have a good transmitting power (Tx) connection to one neighbor and a good receiving connection (Rx) from the other node on the network. Therefore, the routing protocol that this project will use is to discover the best neighbor towards any of its destination. Also the routing protocol will helps with the node to find the best path on the mesh network.

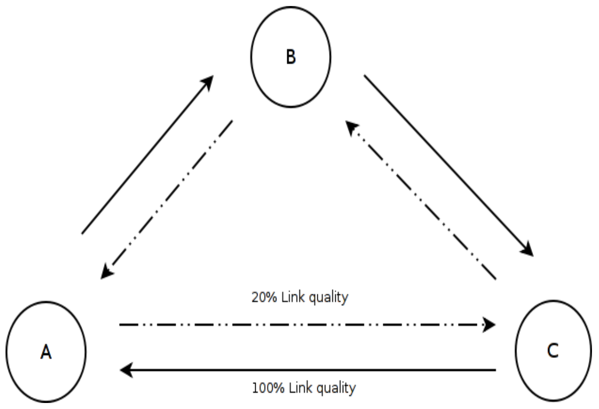


Figure 3: Diagram of Asymmetric routing for Mesh Network

Behaviors of Wireless Mesh Networks.

This project will ensure that the rate used on individual Mesh Potatoes (with each neighbor) will allow easy data communication and better quality of service (wireless links). This project will also allow the network manager to keep track of the expected time to send packet to a remote network on another region (e.g. rural area) during remote accessibility as well as avoid traffic congestion during peak hours.

Chapter 4

USER INTERFACE SPECIFICATION

Introduction

This chapter discusses the design of the system and describes exactly what is expected from the user interface, what it will look like, and how the network manager will interact with the system. The user interface specification for this project will be represented in either a graphical user interface (GUI), command line interface (CLI) or an application programming interfaces (API). We developed a web based application to allow the network manager to monitor and help in the management of the mesh network. To develop and evaluate the software, we constructed a mesh network (Ad-hoc) test bed of our own using the same hardware and software used by the Village Telco (the Mesh Potato routers SECN) [6].

Design Overview of the Project Integration

This project will be designing and developing the frontend and backend application [9, 10] for mesh networks which will mainly focus on network monitoring system (i.e. by frequently checking the network performance, link quality etc.) and on autonomic network configuration. As a future work, we intend to integrate the frontend and backend application into a single one, using the visualization tool as the graphical interface to access all network services or activities.

As we had partially provided in our prototype solution figure 4, our main idea will be to build a complete network management system where all kinds of data stay on the same platform and the network manager decides the type of information to view on the mesh-dash. The frontend application of this project will be aggregating data from the server and presents the information gathered as topology visualization for the network manager to view the activity on the mesh network.

The motivating factor for this topology visualization was that network manager needs to view and see how the nodes on the network are naturally organized as well as their geographical displacement of nodes [11]. That is, the topology will help us to combine the aggregated data and presents them on a richer map for viewing by the network manager, allowing the network manager to quickly access all broader data about the network. For instance, the quality of all links or the number of authenticated users on each node, and with a simple request the network manager could receive more detailed data on specific nodes, such as the names of authenticated users, performance metrics like memory consumption or recent activity history.

The ultimate objective of integrating the frontend and backend application is to pave a new way of visualizing the wireless mesh network whenever they are seen as a scatter graphs under a controlling monitoring and managing system application. In other words, it will enable the network manager to understand and gather needed network information in a faster and easier way than before with a disperse set of tools.

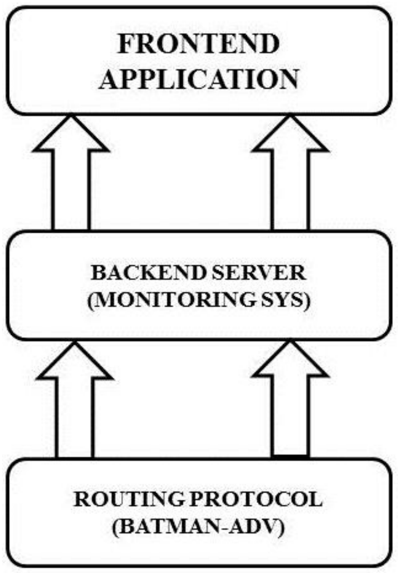

FRONTEND APPLICATION
BACKEND SERVER (MONITORING SYS)
ROUTING PROTOCOL (BATMAN-ADV)

Figure 4: General Overview of the Project Design

Mesh-Mode Wireless Network Design

Figure 5 showed Setup called mesh infrastructure mode. This is an important part of the project, as this is where monitoring component was developed. This design consisted of two Mesh Potatoes and a desktop PC connected to each MP. Each MP was configured with the same subnet mask and different IP addresses to allow packets to be sent between each node. The PCs and the MP formed a wireless ad-hoc mesh network amongst themselves so that a network manager will be able to monitor and visualization the activity of each node in the network. The following figure shows the design of Mesh Potatoes with PCs when building a shell of the user interface for the network manager test bed.

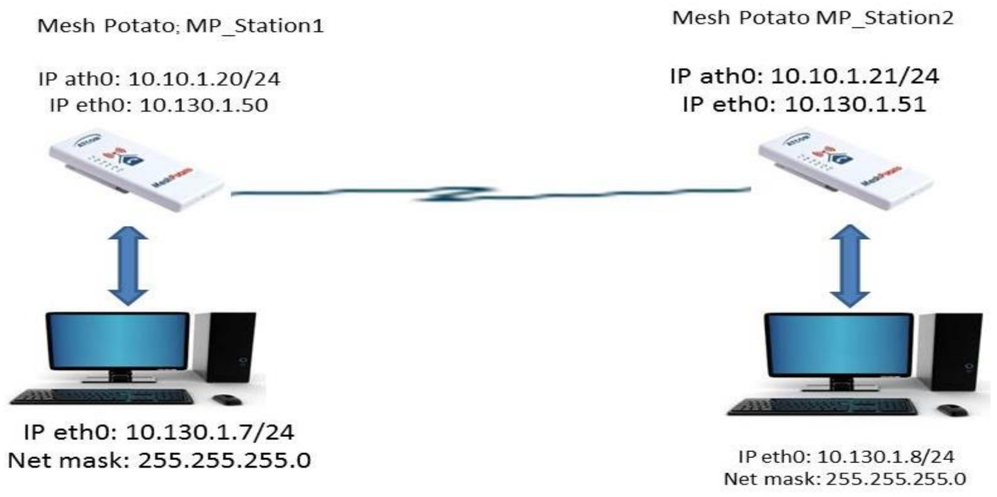


Figure 5: Mesh Potatoes design set-up

Design Interaction between PC and Mesh Potatoes (MP)

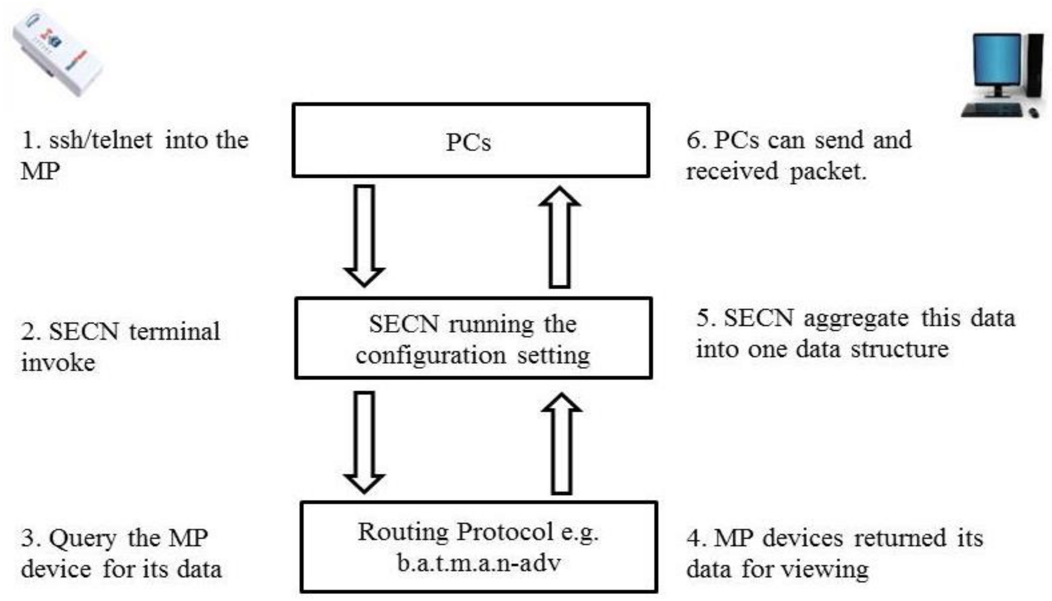

1. ssh/telnet into the MP
PCs
6. PCs can send and received packet.
2. SECN terminal invoke
SECN running the configuration setting
5. SECN aggregate this data into one data structure
3. Query the MP device for its data
Routing Protocol e.g. b.a.t.m.a.n-adv
4. MP devices returned its data for viewing

Figure 6: The design interaction between PCs and Mesh Potatoes

The following figure 6 shows the process of logging into the mesh potatoes using the secure shell (SSH) on the command line interface (CLI). By logging into the MP it will allow the network manager to configure and manage the router for the mesh network. The process of getting into the mesh potatoes is illustrated as follow on the diagram:

Data Collection Interface

The data collection of this backend application design interface will be represented through a command line interface (CLI). A user (Network Manager) will be able to interact with a computer program using command lines. In other words, a user interface in which a network manager type commands instead of choosing from the menu or selecting an icon.

This CLI presents the system user with several options for configuring the router (e.g. Mesh Potatoes) with a simple-to-use interface that meets both needs and wants of the network manager. The reason for using the CLI is that, a network manager will need to configure the mesh potatoes using a secure shell (ssh).

A secure shell is a network protocol that allows data to be exchanged using a secure channel between two networked devices. In addition, using the CLI, a script will be run on the mesh potato to collect data. That data is stored in a file, and its size managed. That data will be transferred to the central server. The server will employ the concept of data mining using extract, transform, and load (ETL) to parses the file to pick out cumulative and snapshot data-points that will be inserted into the database. The database in Figure 5 will be used to automatically populate the graphical user interface (GUI) of the frontend system of this project.

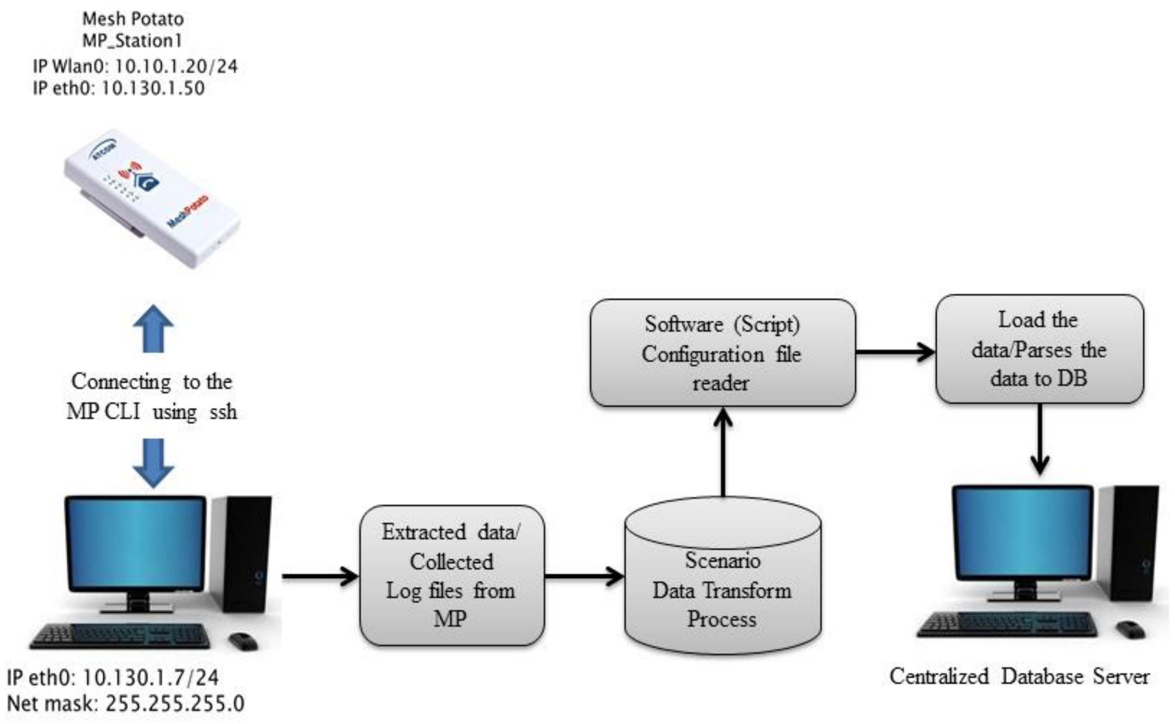
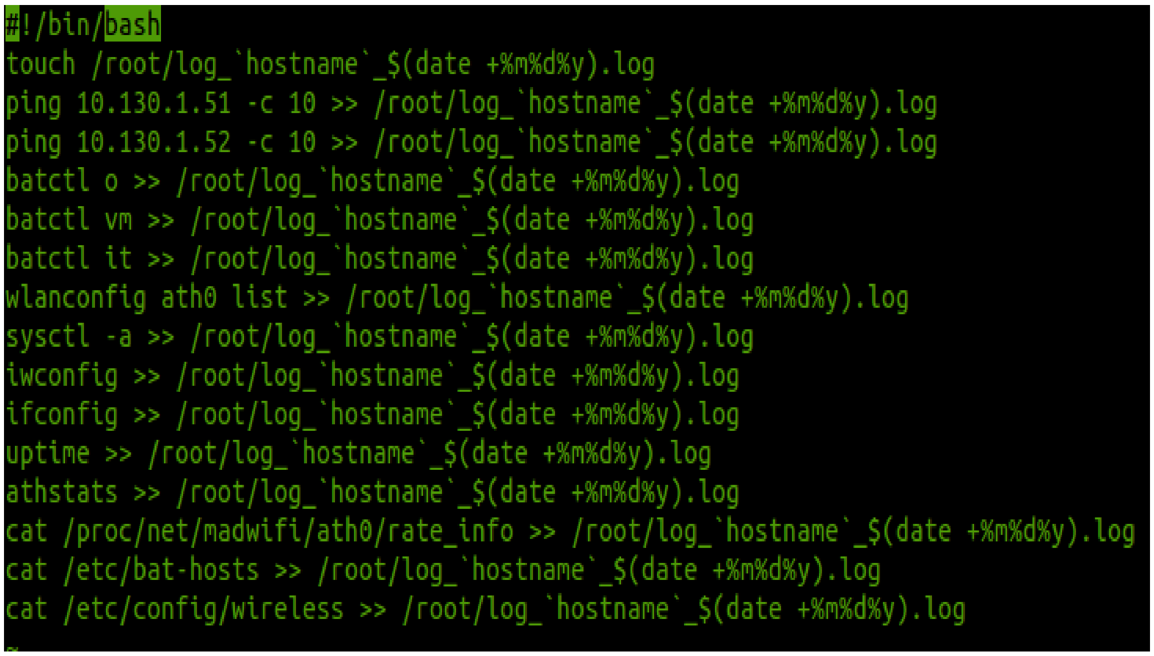

Mesh Potato
MP_Station1
IP Wlan0: 10.10.1.20/24
IP eth0: 10.130.1.50
Connecting to the MP CLI using ssh
IP eth0: 10.130.1.7/24
Net mask: 255.255.255.0
Extracted data/ Collected Log files from MP
Scenario Data Transform Process
Software (Script) Configuration file reader
Load the data/Parses the data to DB
Centralized Database Server

Figure 7: Log on to a client terminal through a PC to log onto the MP.

We then collect the data the log file from the MP for processing, transforming, and loading it into MySQL DB. The following figure 8 shows the screenshot of the executed script using the CLI during the process of collecting the data into a log file. Figure 7 help to shows the illustration of the physical design of the process to collect data from the Mesh Potatoes. The CLI will enable the network manager to run a script for collecting the configuration information (*e.g. ifconfig*) from the mesh potatoes.

By running the script (e.g. executing *batctl o*) on the mesh potatoes, it will gathered some configuration information and stored the output in a log file. The log file will be collected for extraction to pick out cumulative and snapshot data-points that will be inserted into the database. Before parsing the log file, necessary information will be transform using an application file extractor to parse the log file to the database server for populating the frontend GUI.



```
#!/bin/bash
touch /root/log_`hostname`_$(date +%m%d%y).log
ping 10.130.1.51 -c 10 >> /root/log_`hostname`_$(date +%m%d%y).log
ping 10.130.1.52 -c 10 >> /root/log_`hostname`_$(date +%m%d%y).log
batctl o >> /root/log_`hostname`_$(date +%m%d%y).log
batctl vm >> /root/log_`hostname`_$(date +%m%d%y).log
batctl it >> /root/log_`hostname`_$(date +%m%d%y).log
wlanconfig ath0 list >> /root/log_`hostname`_$(date +%m%d%y).log
sysctl -a >> /root/log_`hostname`_$(date +%m%d%y).log
iwconfig >> /root/log_`hostname`_$(date +%m%d%y).log
ifconfig >> /root/log_`hostname`_$(date +%m%d%y).log
uptime >> /root/log_`hostname`_$(date +%m%d%y).log
athstats >> /root/log_`hostname`_$(date +%m%d%y).log
cat /proc/net/madwifi/ath0/rate_info >> /root/log_`hostname`_$(date +%m%d%y).log
cat /etc/bat-hosts >> /root/log_`hostname`_$(date +%m%d%y).log
cat /etc/config/wireless >> /root/log_`hostname`_$(date +%m%d%y).log
```

Figure 8: A shell scripting program for collecting the log files from the Mesh Potatoes

Chapter 5

HIGH LEVEL DESIGN

Introduction

This chapter discusses high level design of the project mainly for the backend monitoring. The high level design for this project will be represented by using several subsystems. The main reason for this is that the project spans across different layers of hardware, system applications and operating systems platforms. Hence, each section in this chapter follows a systematic breakdown of each of these sub components. More so, we will be presenting the object oriented view of the system, analysis of the high level design, and we will describe the objects needed to implement the system. Each one of these objects will be describe and documented, and a data dictionary providing details of each object will be provided.

Design Architecture for Monitoring Mesh Network Only For the Backend App.

The following Figure 9 shows the mesh potatoes architecture view of the backend for monitoring mesh network. This project will discuss each section of the architecture component and explain how each of the smaller modules interacts with each other. This project will be using a script (e.g. Perl script) that will enable easy communication with the network through Village Bus. The Perl script will be located on the server (PC) and it will be divided into smaller modules where each module contains the information needed to communicate with the network. This project further discusses each module as follows:

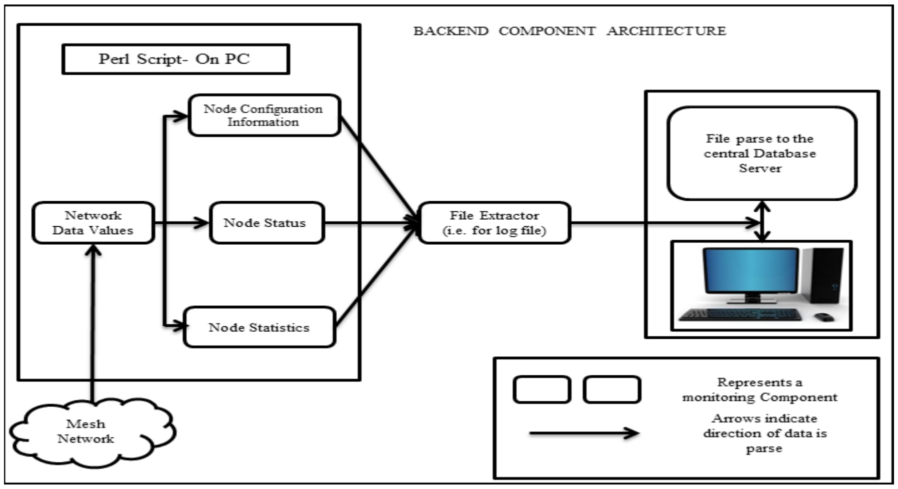

BACKEND COMPONENT ARCHITECTURE
Perl Script- On PC
Node Configuration Information
Network Data Values
Node Status
Node Statistics
File Extractor (i.e. for log file)
File parse to the central Database Server
Mesh Network
Represents a monitoring Component
Arrows indicate direction of data is parse

Figure 9: Mesh Potato Design Architecture for Monitoring Mesh Network

Bash Script Monitoring Component

The bash script is placed on the server, since it is concerned with network communication. Here we will discuss all its modules as follows:

Network Data Values

This module design is the main source of information that will be needed and it will be responsible for collecting data from mesh potatoes and also keep a logfile of the information collected to setup mesh network (i.e. MeshDash), and making it available to other modules component. With this logfile, we will extract or collect the relevant information needed to present to the frontend app of this project for the network manager to visualize. However, some of these outputs are varying every instant (like the information from *wlanconfig* command which gives the RSSI from every neighbor, or *batctl o* which gives the batman metrics to reach every other node in the mesh network). More so, the fluctuated instant are cumulative when executing these commands like; *athstats, uptime, stats from minstrel,* whereas others would remain fixed (i.e. configuration files in */etc/config/* and */etc/bat-hosts*).

Therefore, the idea for the backend was collecting these information's at different moments in time (according to the granularity defined for each of them), keep them in the node parsed and compressed if required/needed, populate them to a centralized server when the traffic in the network was expected to be low (at night) and age them when reception was acknowledged by the server. Note that all the process except the aging will be done using cron for scheduling the script.

Node Configuration Information

This layer will collect information about every mesh node on the network and they will be store on the database server. The node configuration info will contain information for both a wired mesh relay, and wireless 'non-mesh' access point (AP). The wireless interface serves clients as a traditional AP. Furthermore, it will contain the list of all the routers IP addresses and their neighbors IP addresses.

To collect this information we will use the Perl programs to extract the necessary data information and populate the database. The Perl program will enable us to collect the list of each of the routers IP addresses, their byte count and their packet count. Node information like using "*Batctl commands*" which is the configuration and debugging tool for batman-adv gathered information such as MAC addresses of the MP(s) on the mesh.

Other useful commands such as *batctl o* as trans-local or trans-global commands to see the local nodes (mesh only) or all nodes accessible through the mesh (global) are also captured and store in the node config info. Likewise, using commands like */etc/config/batman-adv*, */etc/config/network*, */etc/config/wireless*, enable will us to capture the batman and wireless interface to populate the central database server.

Node Status

This section store the information about links quality, performance testing, client connectivity due to signal attenuation, interference from external devices, and the misbehaving or misconfigured client nodes using batman-adv routing protocol. It will also helps capture connections strenghts of various neighbors in the network.

This section further will show that a single hop IP address for a neighbor in the network might be the best next hop to it when the network manager needed to test quality of link by direct Wi-Fi link. However, the metric that batman-adv is using is not related to RSSI. Batman measures the likelihood that a transmission will reach its destination. We can have a good RSSI value, but awful packet loss. The RSSI value alone does not give an indication that a link is actually working or performing.

Therefore, the node status will help to store the MAC addresses of each MP in the network when checking for mesh potatoes Wi-Fi settings. These can be checked using commands like */etc/config/wireless* or via the GUI. Also we compare *iwconfig ath0* on all of the MPs to *ifconfig ath0*.

To test for connectivity and collect the output even if the MPs interfaces are not configured (i.e. IP addresses have not been configured). We will use these commands like; *batctl tcpdump wlan0*, *batctl 00:09:45:5B:6C:98 ping*, and *batctl traceroute 00:09:45:5B:6C:98* with the MAC address of each MP to gather the node status and store on central database server. We will observe that Ping gives an excellent indication of link quality. In other words, the ping time various greatly when a packet fails to get through, the Wi-Fi protocols are re-transmitted. Hence, re-transmission leads to variable ping times.

Nodes Statistics

This component helps to collect and store data information such as transmission power, node channel, transmission errors, rate-info, BSSID (i.e. is the MAC address of the AP's radio for that service set), and SSID (i.e. is the service set identifier or network name for the basic service set (BSS)). Thus, BSS is a set of stations controlled by a single coordination function on the mesh network.

Data file Extractor App

Figure 10 show the sequence of steps that is executed by the network manager to use this file extractor application. The file extract in Perl Scripting language is started and then various intermediate phases are performed to eventually perform file data collection. The file extractor application will help the mesh network designer (user) to deploy proper configuration layout, access point nodes (AP) and their characteristics.

More so, this design will help the network manager to minimize network infrastructure (mesh nodes and links). The constraints involved will satisfy demands that are placed by Aps (and their underlying networks). Thus, the cost of monitoring and managing mesh network will be by minimizing the mesh nodes and links.

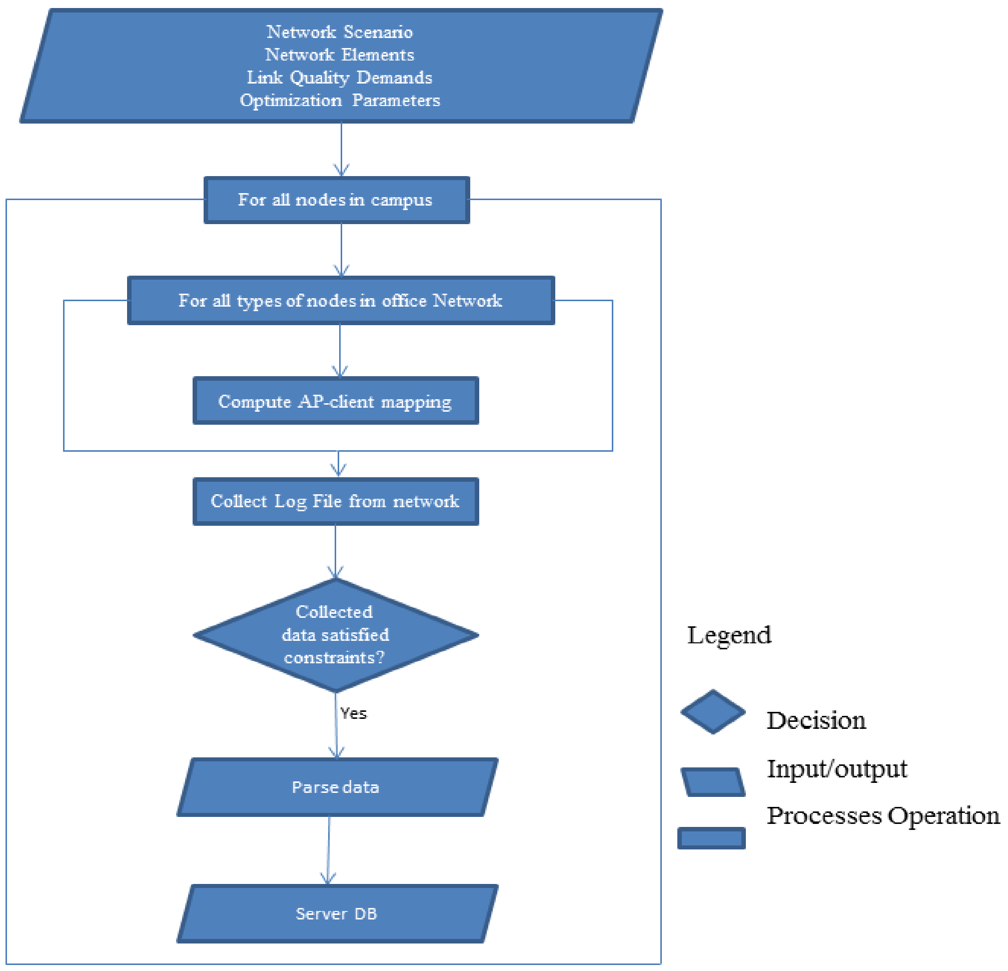


Figure 10: Flow chart representing project file extractor sequence

Furthermore, a set of scenarios for the network manager also needs to be documented as stated in the background part of this project. The chosen tool for this design is the Unified Modeling Language (UML). UML use cases will be used as a representation to the scenario taken place in the mesh network.

Moreover, the UML was chosen due to the fact that it is easy and simple for the network manager to understand the whole concept of the project design. Figure 11 shows how the network manager will interact with the system.

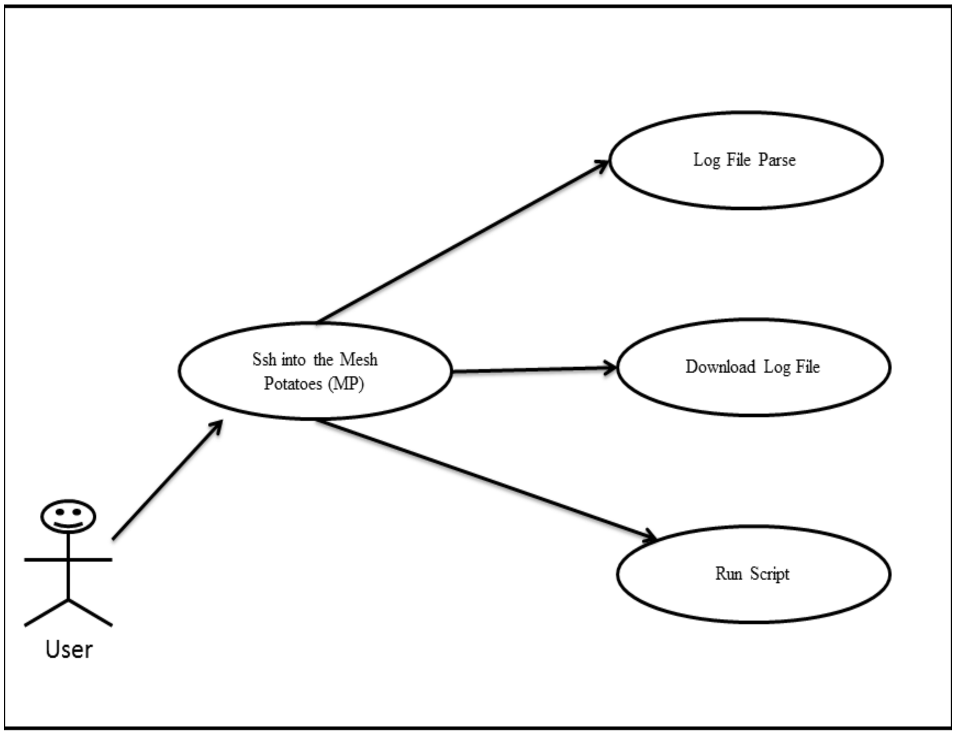


Figure 11: Network Manager Use Cases

Logical Database Architecture for Only the Backend Application

Figure 12 shows the database server architecture. The central database server will be made up of script that takes data from the file extractor, stored it on MySQL database, and presents it to the front-end application (Mesh-Dash), as well as the script that takes node input, process it, and update the database. This database server enables the network manager to know the relationships between the data in the MySQL database.

Hence, this database server uses type method to classify its structure. That is, we have view task and Process task where each task depend on each other input and output relationships. The following is a description of each task expected within the structure of database architecture.

View Tasks

1. Query the database for network information to display
2. Automatically change the input of the node configuration settings on the database server.
3. Present the information to the frontend app for network manager to visualize data.

Processor Tasks

1. Receive HTTP requests from the nodes containing network status information and write that information to the view
2. Generate responses to those HTTP requests based on network configuration settings in the view
3. Receive network configuration data as form input from the view components and write the data to the database.

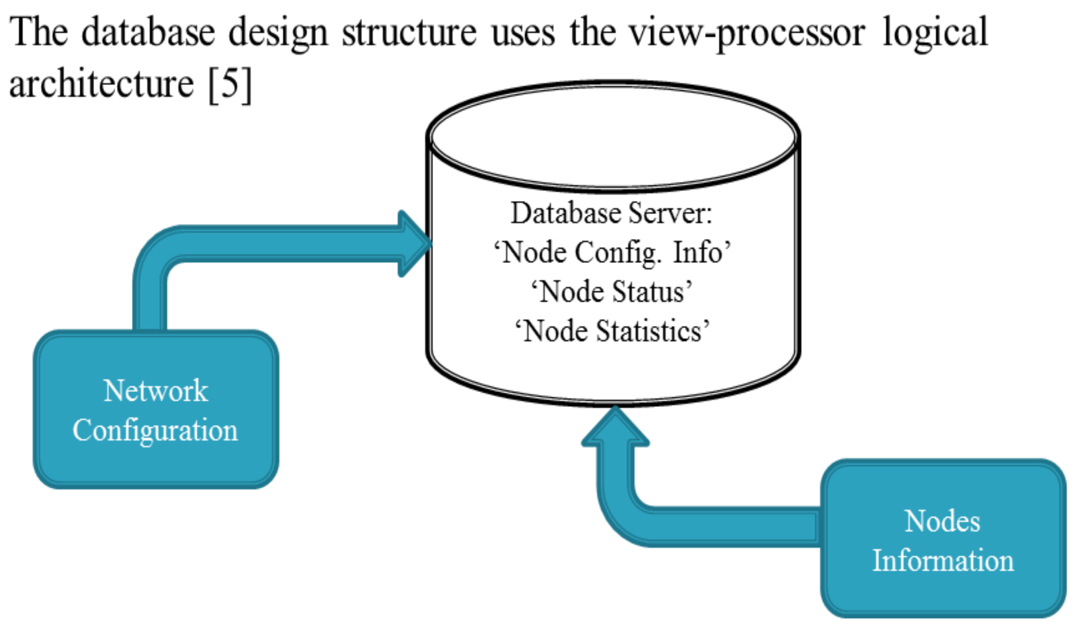

The database design structure uses the view-processor logical architecture [5]

Figure 12: Database server, the logical design architecture for the backend application

Mesh Dash Design Schema

The database structure will be to include the information about the mesh network and the user (network manager) who will be responsible for managing and maintaining the system. That is, data will be contained in the backend application MySQL database. By default we will be using the username "Mesh-Dash" to access the database, with default password "default".

Data Schema

The figure below (Figure 13) shows the entity relationship of the data schema for the proposed system. It consists of the three dimension tables and one fact table. The three dimension tables are node_status, node_config_info and node_neighbours. Each of these tables contains a number of fields and a description of data types.

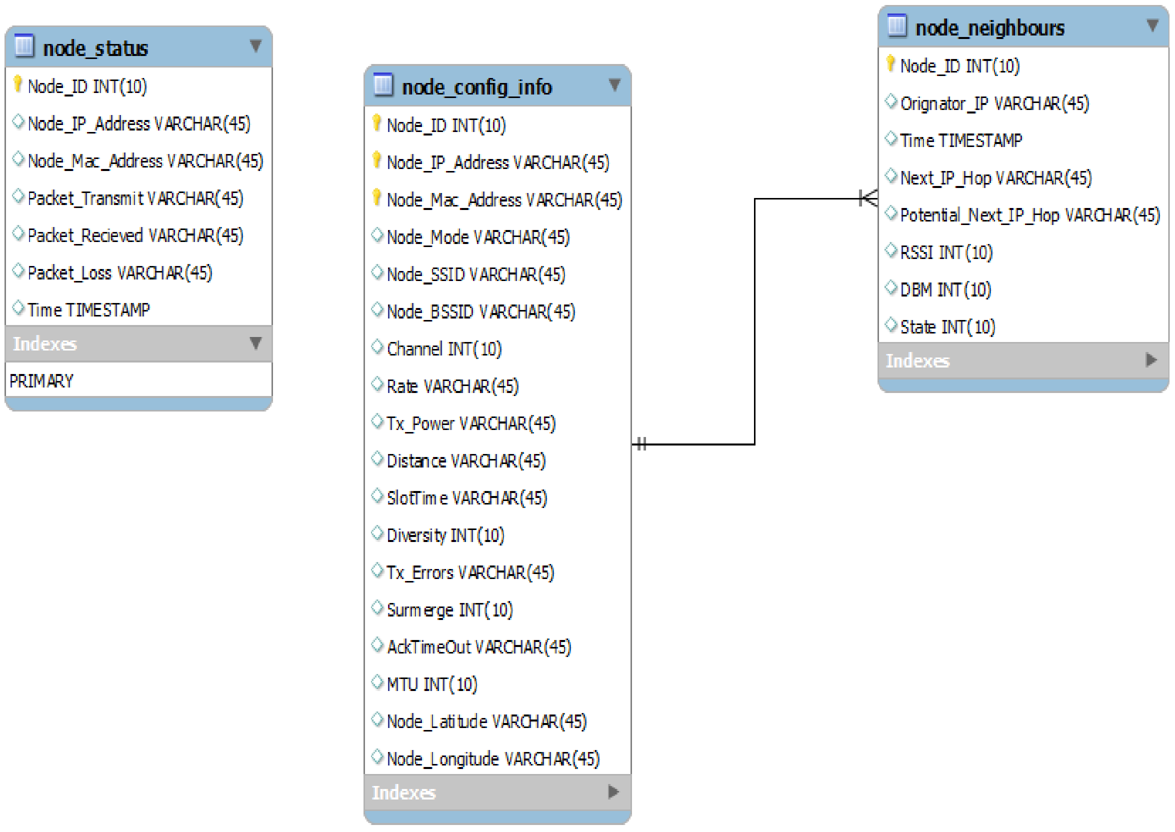


Figure 13: Entity Relationship Diagram for the Mesh Dash Design Data Schema

Chapter 6

LOW LEVEL DESIGN

Introduction

This chapter discusses low level design of the project mainly for the backend monitoring. The low level design for this project will be represented algorithm pseudo code of the subsystems.

Pseudo-code for mesh routing.

```
For all ON/OFF combination of mesh_nodes do
        // on mesh nodes which have been
         Switched ON
         For all ON/OFF combination of links & num_of_mesh_links < max_links do
                          For all demands do
                If demand <  remaining_link_capacity() then
                          Cost   cost_of_shortest_path()   if   cost   <   cost_min   then   cost_min
cosadjust_link_capacity()
                                    End
                          End
                End
```

Network Input Parameters

1. Network elements: Number of AP and potential mesh nodes
2. Network element properties: Properties of nodes and their associated links
3. Network scenario strategy: Properties of deployment layout and node distribution
4. Traffic demands: User generated traffic demands for each AP and clients
5. Link cost functions: Cost functions for fixed and variable transmit powers (TX power) and packet loss.
6. Optimizer parameters and heuristics: Heuristics and initial settings for the optimizer

Script Modules

The following modules are described for the design model of the proposed system for the backend application only.

Node Configuration Information:

Create a network scenario generator that enables to create scenario based on deployment layout, configuration parameters and the number of nodes as well as a scenario to create locations of AP nodes and potential mesh Nodes.

Node Status

For every link constructor uses heuristics to generate list of potential links, an optimization preprocessor that enable us to constructs inputs for optimizer and demand matrix for the constraint file extractor application verifier. The optimizer must invoke to solve MP problem.

Node Statistics

A constraint verifier must be set to verify capacity constraints imposed on scenario by comparing optimizer output with demand matrix. Hence, a topology generator (Mesh Dash) must be constructs to corresponding to the capacity constrained topology. Therefore, the network manager must carry out a simulation by using an external simulator to validate the topology generated in order to visualize what is on the Mesh Dash.

Detailed Data Definitions for the MeshDash Entity Relationship Diagram

The data dictionary contains information describing the content of the Online Registration System. Data is contained in the ‘MeshDash’ MySQL database. The following table gives the detained description of each table contained in the MySQL database.

Name	**Type**	**Description**
Node_ID	Int (10)	Unique network id, primary key (auto-increment)
Node_IP_Address	Varchar (45)	Node IP address
Node_Mac_Address	Varchar (45)	MP Device Mac Address
Packet_Transmit	Varchar (45)	Node transmission power
Packet_Recieved	Varchar (45)	Node transmission received
Packet_Loss	Varchar (45)	Node transmission loss
Time	TIMESTAMP	Give time stamp for network

Table 2: ‘node_status’

Name	**Type**	**Description**
Node_ID	int (10)	Unique network id, primary key and also a foreign key, (auto-increment)
Node_IP_Address	Varchar (45)	Node IP address, a foreign key, and
Node_Mac_Address	Varchar (45)	MP Device Mac Address
Node_Mode	Varchar (45)	Node mode either for client or server

Node_SSID	Varchar (45)	A unique name for the name
Node_BSSID	Varchar (45)	A unique station ID
Channel	int (10)	Signal channel values
Rate	Varchar (45)	Packet rate transmission/ Packet transmission loss
Tx_Power	Varchar (45)	Packet transmission values
Distance	Varchar (45)	Packet distance covered
SlotTime	Varchar (45)	Node interval time
Diversity	int (10)	Distance
Tx_Errors	Varchar (45)	Packet transmission error
Surmerge	int (10)	Submerged distance
AckTimeOut	Varchar (45)	Network acknowledgement time
MTU	int (10)	Message transfer control unit
Node_Latitude	Varchar (45)	Node distance cover on map latitudinal
Node_Longitude	Varchar (45)	Node distance cover on map longitudinal

Table 3: ‘node_Configuration_information’

Name	**Type**	**Description**
Node_ID	Int (10)	Unique network id, primary key and also a foreign key, (auto-increment)
Orignator_IP	Varchar (45)	Source (node) gateway IP address
Time	TIMESTAMP	Time stamp for node transmitting
Next_IP_Hop	Varchar (45)	Next Source (node) gateway IP address
Potential_Next_IP_Hop	Varchar (45)	Nearest Source (node) gateway IP address
RSSI	int (10)	RSSI in Unsigned values
DBM	int (10)	DBM values
State	int (10)	Node state

Table 4: ‘node_neighbours’

User (Network Manager) Design Role (Role-Based Security)

The users will be able to login into the MP and the software application. In this system we will have three types of people to use the system. The role lists is shown in our database indicate that the network manager (detail about the network manager), and remote User (include detail about remote or local user).

Business rules

Figure 14 shows the role based security permission for the design system for both frontend and backend application. However, the following business will be apply to the design model of the

1. There will be only one network manager (user) for a given network; means there can be only one login for network monitoring. He is like a super user for monitoring the network.
2. Network manager (user) can be a local user as well.
3. A single network can contain more than one local users and each local user have different kind of rights/permissions can be configured by network manager.
4. Only remote user allowed per network and rights/permissions can be configured by network manager.

The standard way we will model permissions for the system is through Role-Based Security. The typical model design will looks something like this:

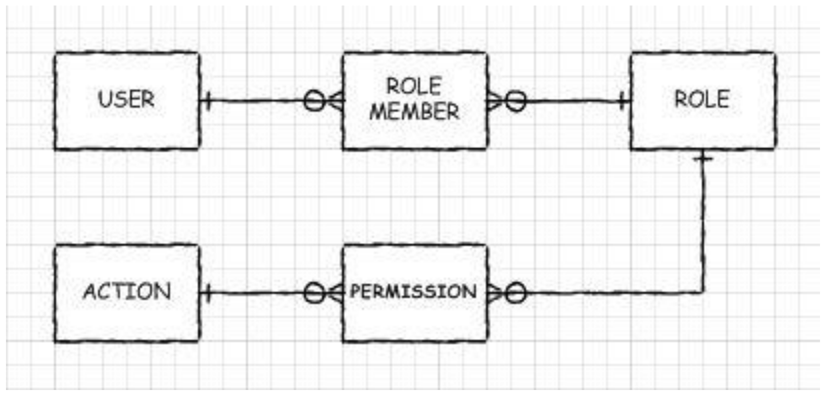


Figure 14: MP Login: The role based security permission

The way it works is you have three tangible things: Users, Allowable Actions (the access that is being controlled) and Roles. Roles are groups of both users and allowable actions, therefore there are two intersection tables which record the people in each role and the actions that role permits. Some people include additional information in the role membership and permission intersection tables, like when the record was created and by whom. Some people keep that kind of audit log information in a separate table.

Chapter 7

IMPLEMENTATION

Introduction

This Chapter describes and explains our project implementation and documented code. Each class and class method used are explain with a one-line description, and a detailed description of the algorithm. The chapter further explains all methods that involve human interaction concerning inputs and outputs. We further noted any caveats, that is, things that can go wrong or things that the code does not address. Hence, the chapter narrows its discussion on a general overview of the project (architecture) of the mesh network monitoring component.

Architectural View of the Project Implementation Using Extract, Transform, and Loading Concepts

In order to facilitate and manage the proposed system we use the general title *Extraction-Transformation-Loading* (ETL) tools (see appendix B for source code concept). To give a general idea of the functionality of these tools we mention their most prominent tasks, which include: (a) the identification of relevant information at the source side; (b) the extraction of this information; (c) the customization and integration of the information coming from multiple sources into a common format; (d) the cleaning of the resulting data set (log file), on the basis of database, and (e) the propagation of the data to the database. Hence, the following figure shows and illustrates how the source system was mapped to the targeted system. Mapping showed the relationship and data flow between the source and target system.

In this report, our source system is Mesh Potatoes which helps to run two shell scripting program *(i.e. Dynamic.sh, and Static.sh)* with a crons for scheduling the script and help to optimize the program at vary time automatically. The Dynamic.sh is a shell program which is responsible for collecting varying network accounting information from the source system; help capture activity on the device and writes into a log file, while Static.sh is a shell program which is only responsible for network values that does not change but remains static and write into a log file. Both shell programs for extracting and analyzing data went through the staging process before we load the required data into MySQL database server. We referred to the whole processes as ETL concepts which have integration with the frontend application.

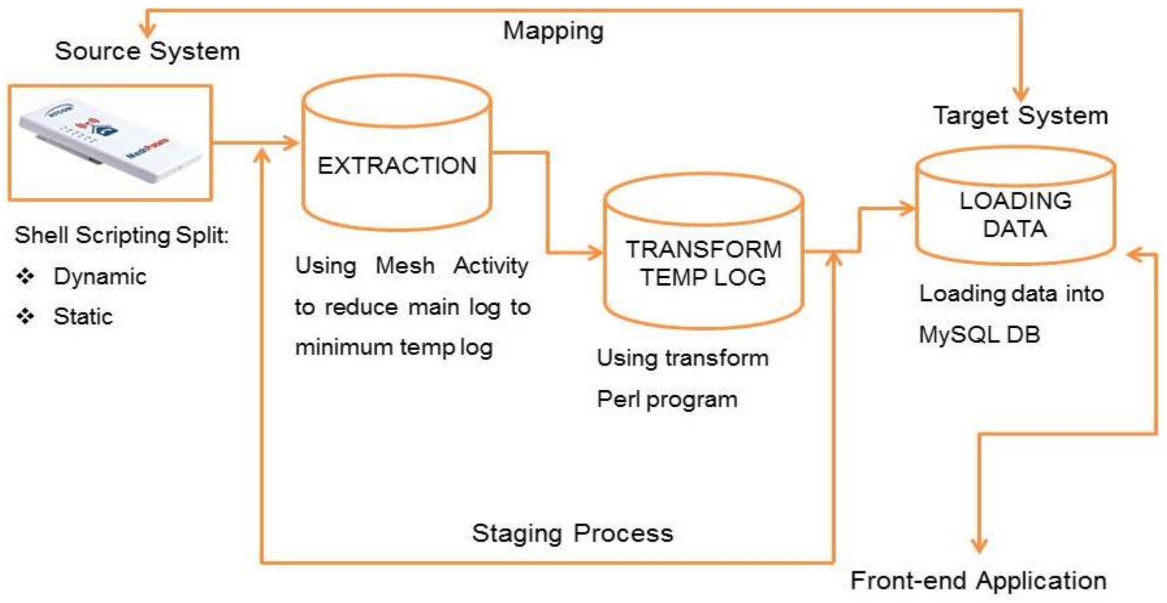


Figure 15: General Overview of the Project Implementation

Structured Program Implementation

This section include code layout, the style used, and technologies employed (e.g. REGEX in Perl language, and with Database Management in Perl). We design a meta-scheduler [an entity which manages the mesh resources and activity to make the network usage easier for the network manager. Example of such meta-scheduler is the Mesh-Crontab]. Since network manager usually send a job to the target system, and from that moment on, it will take care of any interaction with the mesh-network to get the job executed (such as selection to resources, data transfers , authentication issues, job monitoring and migration's).

Further we make use of the Crontab commands, found in UNIX and Unix-like operating systems, for timing and meta-scheduling the following programs to be executed periodically: See Appendix E for source code.

1. Perl program to extract and transform mesh activity from source system and load it to the targeted system e.g. Database server.
2. Dynamic and Static Scripting programs

The reason for setting up crontab is to minimize job's completion time and execution costs, minimizing data movement, and maximizing system throughput and resources utilization. We set cron in Mesh Potatoes to start automatically from */etc/crontab/init.d* on entering multi-user run levels. Cron searches it spool area (*/var/spool/cron/crontabs*) for crontabs files (which are named after accounts in *etc/passwd*). Crontab found are loaded into memory.

Therefore, cron was to help automates system maintenance or administration—though its general-purpose nature makes it useful for things like connecting to the Internet and downloading the log file at regular intervals.

File Redirection and Pipes Implementation

Reading and writing in both shell and Perl language work exactly like getting input from or sending output to the user, but with the standard input redirected to come from a file or with the standard output redirected to a file. The shell program in this case is our source system (Mesh Potatoes encompass both the Dynamic and Static.sh) which allows file separation for scheduling purposes on the Mesh Potatoes:

We therefore separated: parameters that changes dynamically (include *batctl o*, *wlanconfig ath0 list*, *rate-info* etc.) from others parameter that do so in an aggregated way include *athstats*, *ifconfig*, *acktimeout*, *slot-time*, *tx-power* etc. More so, the shell program in our source system allows us to; See Appendix C and D for source code

1. Check if file exists
2. If not we create the file
3. Else we open the file to edit (Open to read from it)
4. Go in a while loop
5. Reading from another files
6. Write to the end of the file (Append to file but end of file), then we close and exit the file.

Most importantly, we realize that the true power of shell scripting lies not in the scripts themselves, but in the ability to read and write files and chain multiples programs together in interesting ways.

Therefore, the program in our UNIX-based or UNIX-like (Ubuntu) system has three basic file descriptor normally a reference to a file or socket reserved for basic input and output (often abbreviated STDIN), standard output (STDOUT), and standard error (STDERR).

Implementation of MySQL Database with the Perl Program

Perl script using the DBI methods has become very easy to write database application using DBI. DBI stands for Database Independent Interface for Perl which means DBI provides an abstraction layer between our Perl source code and the underlying database, allowing us to switch database implementations really easily. DBI is independent of any database available in backend. The DBI is a database access module for the Perl programming language. It defines a set of methods, variables, and conventions that provide a consistent database interface, independent of the actual database being used. The following diagram show DBI, DBD::ODBC architecture where its define the actual DBI application programming interface (API), the route method calls for the appropriate drivers, and the DBI driver provides the various supports services to them. The driver implemented in this case was the DBD:: MySQL which actually perform the operations on the database.

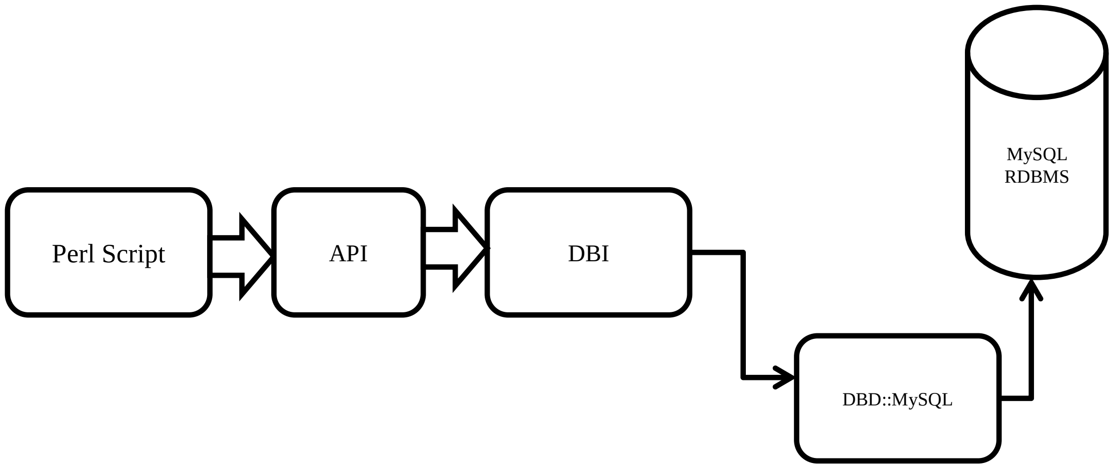


Figure 16: MySQL Database Integration with the Perl Program

For the Perl, we used Perl 5.8 but we only need the minimum required by the DBI and DBD::ODBC modules which is currently 5.6. We use *Perl --version* to see what version of Perl we installed. We also used DBI 1.45 for the DBI module. To see if we have a recent enough version of DBI installed we run *Perl -e 'use DBI 1.40* on the Ubuntu terminal.

Database Management Notation and Conventions

The following are notation and conventions we used in our project. Most importantly, we follow the normal standard for naming convention and methods notations. The notation and conventions includes: Also see Appendix B for code implementation

1. $dsn: This represent Database source name
2. $dbh: This represent Database handle object
3. $h: This includes any of the handle types above ($dsn,$sth, or $drh)
4. $rc : General return code (Boolean: true = ok, false = error)
5. $rv : General return value (Typically an integers)
6. @arr: List of values returned from the database
7. $rows: Number of rows processed (if available, else - 1)
8. $fh : A filehandle
9. Undef : Null values are represented by undefined values in Perl
10. \%attr Reference to a hash of attribute values passed to methods

The reason for the notations and conventions is to avoid inconsistency and prevent us from having modules redundancy. Since this project is dealing with several log files. This project further make use of syntax which defines a set of rules that enable us to combine regular expression symbols which we considered to be a correctly structured fragment in Perl language. This applies both to Shell and Perl programming language, where our document represents source code, and log file where the document represents data. Generally, syntax of a language defines its surface form [17].

Therefore, our Perl language with MySQL database is text-based computer programming which is based on sequences of characters. We used the regular expression to extract the sequence of characters. Documents that are syntactically invalid are said to have a syntax error.

Main Database Operations

MySQL database support SQL transactions. This means that the network manager can make a whole bunch of queries which would modify the MeshDashDB database, but none of the changes are actually made. Then at the end you issue the special MySQL query commit, and all the changes are made simultaneously. Alternatively, we issue the query rollback, in which case all the changes are thrown away and MeshDashDB database remains unchanged. Perl DBI module provides *begin work* API, which enables transactions (by turning *AutoCommit off*) until the next call to commit or rollback. After the next commit or rollback, *AutoCommit* will automatically be turned on again. The following operations and method calls are discussed below. We implemented and executed each method in our main program for the backend monitoring application program for the network activities. Kindly note that for each method called it will help to reduce processes time and saved system resources. Commit is the operation which gives a green signal to database to finalize the changes and after this operation no change can be reverted to its original position. If we are not satisfied with all the changes or we encountered an error in between of any operation, we can revert those changes by using rollback API.

Therefore, if the transactions are simple, we save ourselves from the trouble of having to issue a lot of commits. When we make the connect call, we specify an *AutoCommit* option which will perform an automatic commit operation after every successful query.

Read Operation

This method enables us to fetch useful information from the MySQL database. So once or database connection is established, we are ready to make a query into the MySQL database. The following is the procedure we employed to query all the redundant data. Also see Appendix B for code implementation.

1. We prepared the SQL query statement based on the required conditions. This was done using *prepare ( )* API
2. We executed SQL query to select all the result from the database. This was done using execute *executing ( )* API

3. We fetched the entire results one by one and proving those results. This was also done using *fetchrow_array ( )* API. The fetchrow_array method returns the values of the next row in the result set as a list, which we assigned to an array. The order of the elements is as the order of the fields in the query (*Packet_transmitted*, *Packet_loss* in our case).
4. We released the statement handle. This was done using *finish ( )* API.

Update Operation

This method in the source code enables us to update on the MySQL database, that is, we update one or more records already available in the database tables. Following is the procedure we used to update all the records in the MeshDashDB database. This will take three steps: (Also see Appendix B for code implementation)

1. Preparing SQL query based on required conditions. This will be done using *prepare ( )* API.

2. Executing SQL query to select all the results from the database. This will be done using *execute ( )* API.

3. Releasing Statement handle. This will be done using *finish**( )*** API

If everything goes fine then commit the operation otherwise we rollback the complete transaction. See next section for commit and rollback APIs.

Insertion Operation

This method is required when we want to create our captured data into MeshDashDB. The backend application followed the procedure to create single records into MeshDahDB. More so, we created many records in similar fashion. Record creation takes the following steps: (Also see Appendix B for code implementation)

1. We prepared SQL statement with INSERT statement. This was done using the *prepare ( )* API
2. We executed the SQL query to select all the results from the database to verify if we are storing the right information into MySQL database. This was also done using *executing ( )* API.
3. We released the statement handle. This was also done using the *finish ( )* API
4. We committed the operation if everything goes well, otherwise we rollback the complete transaction.

Delete Operation

This method enables is required when the network manager want to delete some records from the database. Following is the procedure to delete all the records from MeshDashDB database. This operation will take following steps. (Also see Appendix B for code implementation)

1. Preparing SQL query based on required conditions. This will be done using *prepare ( )* API.
2. Executing SQL query to delete required records from the database. This will be done using *execute ( )* API.
3. Releasing Statement handle. This will be done using *finish ( )* API

If everything goes fine then commit the operation otherwise we rollback the complete transaction.

Common Regular Expression Used

The following three Perl regular expressions were used in implementing the backend application. The most commons are: (See Appendix B for the use of regex)

1. The Match Regular expression (~m//)
2. Substitute regular expression (~s///)
3. Trans literal regular expression (~tr///)

Kindly note that the forward slashes in each case act as delimiters for the regular expression (regex) that are specifying. We further make use of the data types (that is, Scalars and Arrays) in the Perl language to do pattern matching either to match a string or statement to a regular expression.

Chapter 8

TESTING

Introduction

The previous chapter focused on the implementation of the backend application. It gave a thorough documentation of the code used and explained the contribution of each part. How each part works and make the application practical and functional. More so, the chapter explained how each of the application components has contributed to the development of the project. Therefore, this chapter focuses on testing criteria and analysis where different testing strategies are used. The chapter explained in detail the security testing, unit testing, backup and recovery testing, and performance testing for the application system. We then evaluate the analysis and show the results. The process of testing we document it for both the user (application itself) and the system (operating system), to ensure that the solution meets the system requirements and that it is scalable, robust, and efficiency.

Testing Strategies

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under the test [19]. We test this application to uncover errors that were made inadvertently as it was designed and constructed [20]. This testing will be conducted by the Network manager and the developer of the application. We created a testing specification document which defines the plans that describes an overall strategy and a procedure that defines specific testing steps and the tests that will be conducted.

Now that the application is up and running, the kind of things we will need to test in the application will include: checking all the nodes on the network runs the same batman-adv version, checking and test the quality of the data in the application database (i.e. these includes measures in the fact table and data in the dimension tables), and testing the script for backing up the database (e.g. meshDashDB.sql). However, one doesn’t need to test transactions as this is the responsibility of the ETL concept discussed in chapter 7 of this report. To prepare for the test, we set up four Mesh Potatoes (i.e. source system are up and running and can ping each other), and a target system (i.e. server running open source operating system e.g. Ubuntu Server OS) running an instance of the backend application. Then we created a real time mesh network to automate the collection of the logs from the source system and also remotely transferring the logs to the target system to extract and analyze.

Further, we time the cron job scheduler to collect the activities on the network and execute the backend application program on the target system to populate the MySQL database. Creating a password to protect the system allows non-local user to connect to the server and also protect the system from outside threat. Thus, the database was populated by adding more records and made MySQL database to listen to system source system via SCP listen addresses configuration parameter. There are two different times that we need to test our data with. We need to test it before our ETL load and also after [21]. We can then run the regular or standard ETL process into the fact or dimension table and then re-run the test with the new expected results. These two sets of tests are to be run on known and static data.

Therefore, we describe the test plans, approach used or procedure to take, and the different tests that will be conducted. Thus, this chapter of the documentation explains and uses different testing strategies. These testing strategies include:

Performance Testing

This section of this chapter focused on testing the performance (i.e. execution time) of the application. We install the application on the target server to manage and monitor the mesh network. We compare the application running timing to the operating system (OS) timing running on the device such Mesh Potatoes. Further, we compare the central processing unit (CPU) utilization and the paging size (e.g. space occupied on the system memory) that the application will consume when executing it on the target server (e.g. personal computer). We then use a graph to display the timing effect and report on the number of instructions (N) to be executed. We will assign the number of instructions (N) to execute the application with the CPU clock cycles (e.g. average clock per instructions).

This performance testing (such as execution time and speed of the application) will address one of the testing issues or challenges that the application will encounter while executing the application with the mesh Potatoes. Hence, we will be expecting that the application should utilize less CPU and memory space during the process of monitoring the mesh network activities.

Methodology: Execution Time, and Its Short Overview

Execution time of this application depends not only on the processor frequency but may also depend on many other factors like main memory speed and I/O speed. Thus, execution time of this application is the amount of time it takes the program to execute in seconds. Time (i.e. computers does several tasks e.g. elapsed time based on a normal clock, CPU time). Computer time is time spent executing this program. Although the performance testing of this application will be differs on several computer resources.

For instance, if computer program utilize too much of computer resource and no proper care is taken into consideration, such program will crash the network and then leads to serious issues. However, we will conduct performance testing to help prepare for unforeseen circumstance and also help to improve the system resources usage.

That is, performance testing will help us to improve speed of the application when using it for monitoring the mesh network. Execution time of this application can be divided into two parts.

1. User time is spent running the application program itself.
2. System time is when the application calls operating system

The following figure illustrate how the User time and System time are related. They both summed up to make up the execution time of the application.

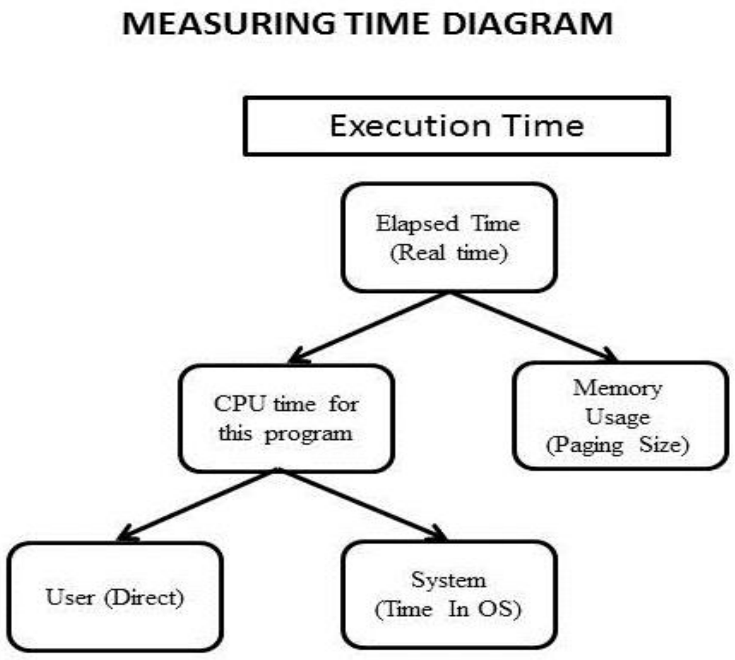


Figure 17: Execution Snap-Shot

The figure above shows the overview of how will be conducting the performance testing. We will then carry out the performance testing in general, particularly on the User time to System time. Further, we will start by comparing the mesh application time to operating system resources time, and also server CPU utilization to paging size.

Thus, our application on the target computer will run according to a clock cycle that runs at a steady rate and time. The execution time of the application will clearly depends on the number of instructions; however difference instructions take difference times.

Why we Tested Performance

We tested the application to provide an acceptable response time over the CPU utilization. The time collecting the log file and running the application on a server is needed to be tested, that is, it must be very fast and take less time to run. Performance tests are often coupled usually both the hardware and the application software itself. In other words, it is often necessary to measure resource utilization (e.g. processor cycles) in an exacting fashion [20].

Performance Testing Representation: Calculating the Measuring Time

The following formula will be used for calculating the execution time (i.e. both User time and System time). Theoretical; we will calculate the performance and speed of the application to operating system. That is, by finding the execution time of the application to the target computer (e.g. Server) operating system. The following figure 18 diagram show how we will carry out the performance comparison: Mathematically, we will represent our calculation as shown in the figure below [18]:

$$\text{Performance (X)} = \frac{1}{\text{Execution (X)}}$$

$$\frac{\text{Performance (X)}}{\text{Performance (Y)}} = \frac{\text{Performance (Y)}}{\text{Performance (X)}} = N$$

$$\text{CPU clock cycle} = N * \text{CPI}$$

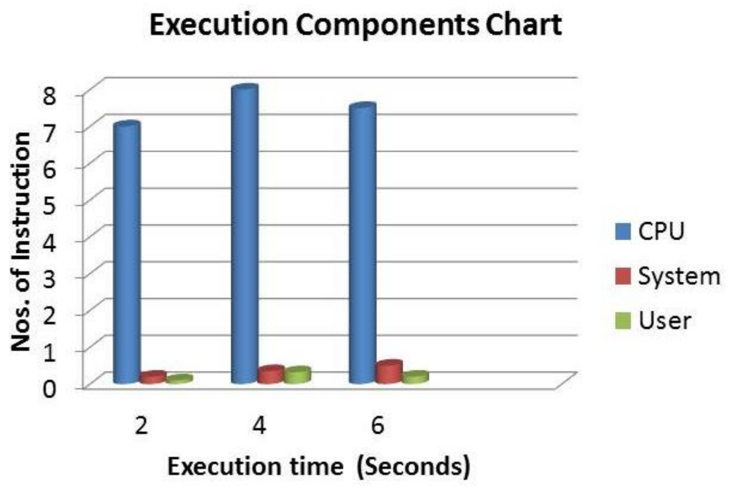
$$\text{Execution time} = \text{CPI} * \text{Cycle time} * N$$

Please note that:
X represent that application time and,
Y represent the system time.
N = Number of Instructions
CPI = Average clock cycles per instructions

Therefore, X will be N times faster that Y.

Figure 18: Mathematical Representation of Execution Time

Graphically, we will present the result on a graph as shown in figure 19 in the chart, and thus we display and represent the different scenario for the comparison.



Notes:
Cumulative Sum (System time) = 0.51 sec
Average per Instructions = 0.0085 sec

Cumulative Sum (User time) = 4.5 sec
Average per Instructions = 0.75 sec

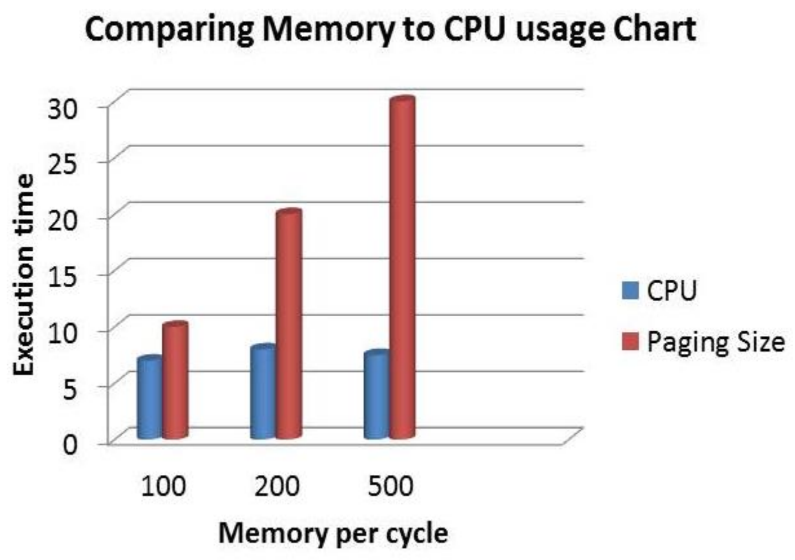
Execution time = CPI * cycle time * Nos of Instruction
Note:
CPI = CPU per Instructions

Figure 19: Graphical Representation of Execution Time

(Use time and System time versus CPU)

The figure shows the execution time for both the User time and System time and compares the result to the CPU utilization of the target system. Further, we will compare the entire time to the target system memory space (i.e. paging size). The execution time of the application program clearly must depend on the number of instructions; however difference instructions take difference times. Particularly, we find the maximum, minimum, and average with and without load of the backend application (i.e. no other program on the background).

Furthermore, figure 20 shows the result comparison after comparing the entire time with the target system memory space (i.e. paging size). Therefore, we present the statistic of the result on figure 20 as follows.



Total Maximum of CPU time = 5570 clock cycles
CPU Average per Instructions = 557 sec
Average Execution Time = 92.83 sec

Maximum Paging Size Usage = 332772 (bytes)
Average per size = 5546.2 bytes

CPU Usage = (User + System)/ Total Nos. of Instr.

Figure 20 Graphical Representation of Execution Time

(CPU versus Paging Size)

The graph above shows the statistic results when comparing the execution time (e.g. CPU utilization) of the application to the memory space during performance testing. Performance testing of this backend application was carried out live to test the run-time of the software within the context of an integrated system. Thus, the execution time of this application is the amount of time running the application to execute in seconds. By our calculation of the time took to execute the program, the application was executed 10 times during which we test for the performance of the CPU and the application itself. This was done and measured in clock cycle per seconds.

From the above performance testing analysis and illustration, we concluded that the mesh application utilize few system resources with better execution time as well as the time it took us to logon and collect the data log on the Mesh potatoes for extraction, transforming and loading. Hence, this mesh application will require the network/system administrator to dedicate a good and fast CPU computer to run the program with no background application except mesh application itself.

Unit Testing

We will conduct unit tests on the internal processing logic of the program. The following logic will be highlighted:

1. Functions: We ensure information flow properly into and out of the program unit under test.
2. All program including errors handling paths are checked and tested.

To optimize our scripting application code, we will conduct the unit tests to help identify potential bottlenecks on our program codes.

Further, we will conduct unit tests to check that we inserted the right data into the database for the backend application usage. We will age data based on external input (i.e. an SQL Injection Attacks will be avoided) program method. This section of the application program is fragile and error prone, that is, both codes profiling and benchmarking are amongst other basic debugging tools we will use for our testing procedure.

Methods

Unit tests were conducted to know if the application database integrates properly with the Application Programming Interface (a.k.a. API) and we ensure that this integration is effective and efficient. Hence, the unit tests help us to increase database-to-program communication efficiency. Perl DBI modules will protect our scripting application from MySQL injection attacks.

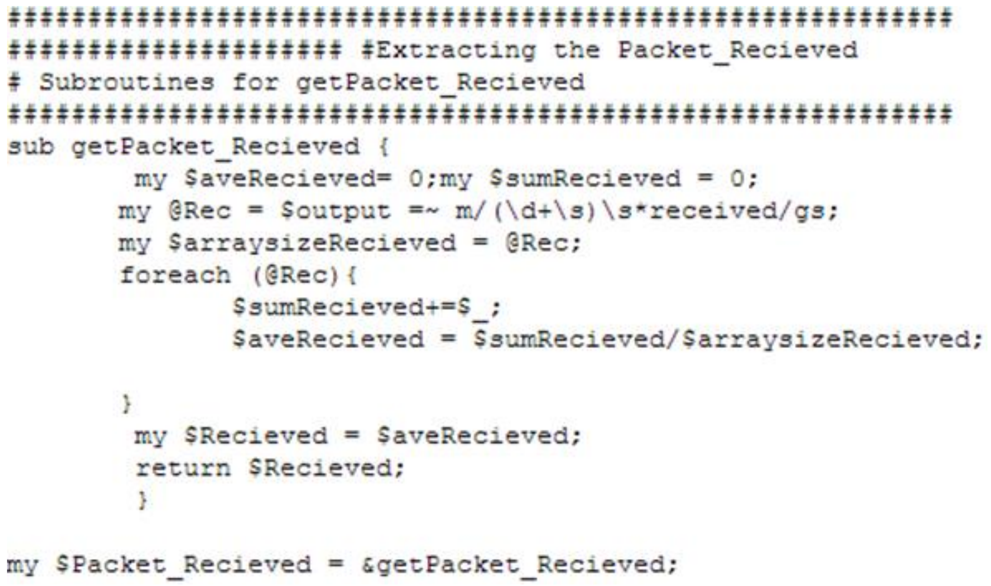
One important benefit of conducting unit testing is that it will give us the confidence that our code works as we expect it to work. Unit tests will give us the self-assurance to do long term development because with unit tests in place we will know that our foundation code is dependable each Perl modules. Unit tests give us the assurance to refactor our code to make it cleaner and more efficient. Unit tests also save us time because unit tests will help us prevent regressions from being acquaint with and unconstrained when we will be conducting performance testing. Once a bug is found, we will write a unit test for it, we will fix the bug, and the bug will never make it to production staging again because our unit tests will catch it in the future.

Another benefit is that unit tests will provide excellent inherent documentation because unit tests will show exactly how our code is design and will be use. Thus we believe so strongly in the value of unit tests that we will write for our program code. The following are the unit tests structure and criteria we will be using to conduct our testing:

Unit Test Structure

We use the following structure for the unit testing. We ensure that the unit tests follow this basic structure:

1. We set up conditions for our unit testing: Our script program methods declaration will perform some sort of operation upon data collected from the Mesh Potatoes. So in order for us to test our methods, we will need to set up the data required by each method in our program. That is we will declare our variables as well as setting up the method to capture right data into our MySQL database. For instance, our unit test looks like this in figure:



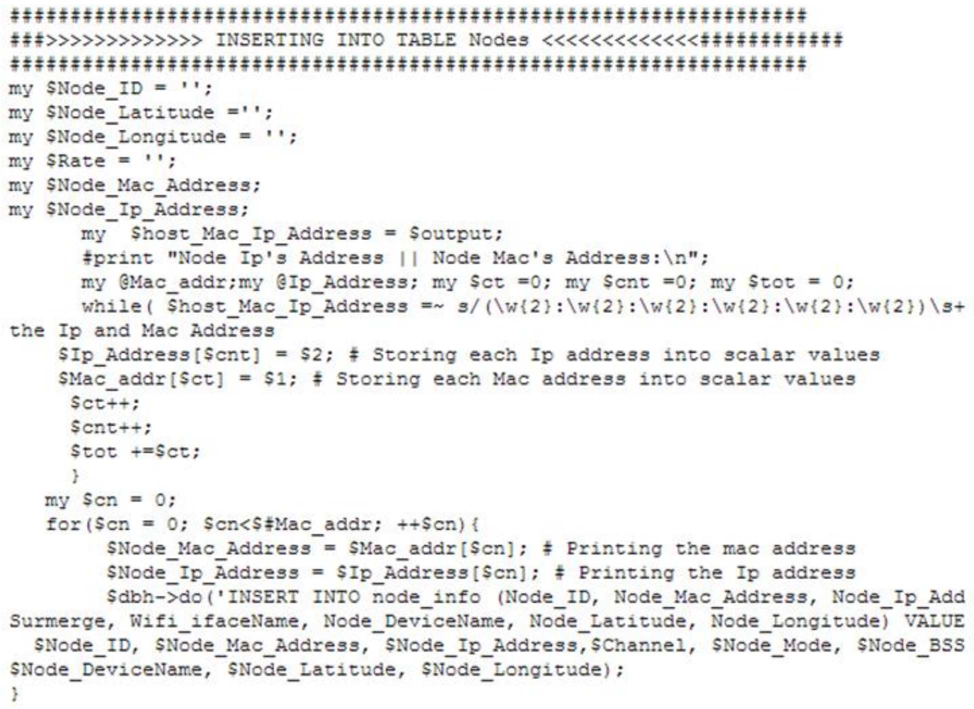
```
############################################################
##################### #Extracting the Packet_Recieved
# Subroutines for getPacket_Recieved
############################################################
sub getPacket_Recieved {
         my $aveRecieved= 0;my $sumRecieved = 0;
        my @Rec = $output =~ m/(\d+\s)\s*received/gs;
        my $arraysizeRecieved = @Rec;
        foreach (@Rec){
                $sumRecieved+=$_;
                $aveRecieved = $sumRecieved/$arraysizeRecieved;

        }
         my $Recieved = $aveRecieved;
         return $Recieved;
         }

my $Packet_Recieved = &getPacket_Recieved;
```

Figure 21: Setting Up Condition for Our Unit Tests

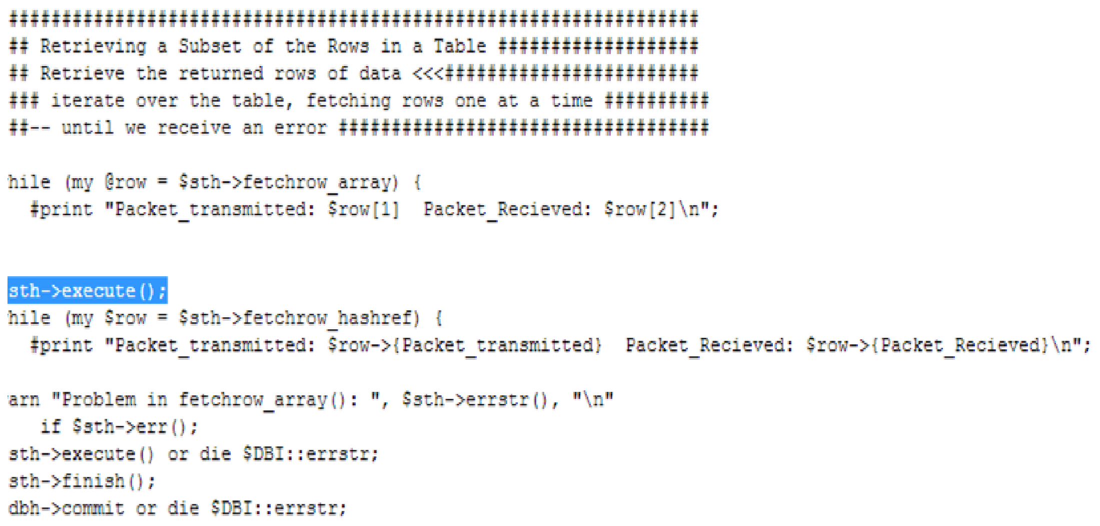
2. We call our program method or Trigger and we ensure that they were being tested: Once we set up the appropriate input data, we execute our code. Testing a method requires us to call the method directly. In our case, we tested a trigger method called insert function, so we perform the action that causes the trigger to execute. For example in the figure below, we insert the following information into our MySQL database for the backend monitoring as shown below:



```
##################################################################
###>>>>>>>>>>>>> INSERTING INTO TABLE Nodes <<<<<<<<<<<<############
##################################################################
my $Node_ID = '';
my $Node_Latitude ='';
my $Node_Longitude = '';
my $Rate = '';
my $Node_Mac_Address;
my $Node_Ip_Address;
      my  $host_Mac_Ip_Address = $output;
      #print "Node Ip's Address || Node Mac's Address:\n";
      my @Mac_addr;my @Ip_Address; my $ct =0; my $cnt =0; my $tot = 0;
      while( $host_Mac_Ip_Address =~ s/(\w{2}:\w{2}:\w{2}:\w{2}:\w{2}:\w{2})\s+
the Ip and Mac Address
    $Ip_Address[$cnt] = $2; # Storing each Ip address into scalar values
    $Mac_addr[$ct] = $1; # Storing each Mac address into scalar values
     $ct++;
     $cnt++;
     $tot +=$ct;
     }
   my $cn = 0;
   for($cn = 0; $cn<$#Mac_addr; ++$cn){
        $Node_Mac_Address = $Mac_addr[$cn]; # Printing the mac address
        $Node_Ip_Address = $Ip_Address[$cn]; # Printing the Ip address
        $dbh->do('INSERT INTO node_info (Node_ID, Node_Mac_Address, Node_Ip_Add
Surmerge, Wifi_ifaceName, Node_DeviceName, Node_Latitude, Node_Longitude) VALUE
  $Node_ID, $Node_Mac_Address, $Node_Ip_Address,$Channel, $Node_Mode, $Node_BSS
$Node_DeviceName, $Node_Latitude, $Node_Longitude);
}
```

Figure 22: Call Program Method or Trigger

3. We verify that the results or return were correct: We verify that our code works as we expect it to work. If this verification fails it will show that our Unit tests do not work hence we will check and correct the code again. A good way we confirm that our unit tests is properly verifying results is to use my $sth->execute () methods. If this does not work, then our tests do not verify results properly. For example our code for verifying code for trigger look like this in the following figure:



```
#################################################################
## Retrieving a Subset of the Rows in a Table ##################
## Retrieve the returned rows of data <<<#######################
### iterate over the table, fetching rows one at a time ##########
##-- until we receive an error ##################################

hile (my @row = $sth->fetchrow_array) {
  #print "Packet_transmitted: $row[1]  Packet_Recieved: $row[2]\n";


sth->execute();
hile (my $row = $sth->fetchrow_hashref) {
  #print "Packet_transmitted: $row->{Packet_transmitted}  Packet_Recieved: $row->{Packet_Recieved}\n";

arn "Problem in fetchrow_array(): ", $sth->errstr(), "\n"
   if $sth->err();
sth->execute() or die $DBI::errstr;
sth->finish();
dbh->commit or die $DBI::errstr;
```

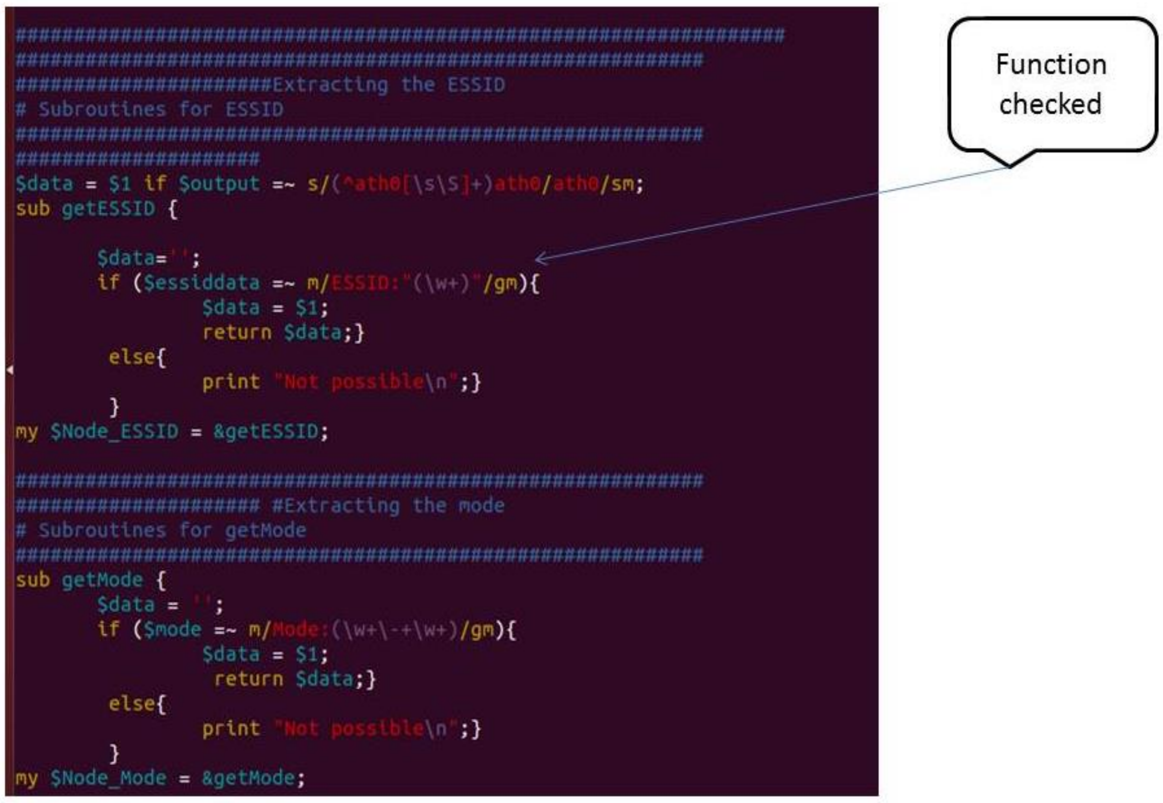
Figure 23: Code Verification for Insert Methods

4. We clean up modified records: We clean up our record, which are the results of trigger executions that affect existing data. However, we will only commit right information when we get the code right and put that right information into the database. We will insert, delete, and modify records without having to write any code that will clean up our changes.

Our subroutine functions will be put on same files which only contain code for insert, update, and delete definition, so these functions are located in the same subroutine class file. The reason for same file is because Perl DBI integration works well when using all of these functions on a role. With our subroutines classes we will always have the option to change the internal implementation of our code should the need arise.

What We Test

Broadly speaking, we will test our code and database logic. How thoroughly we will test that logic will probably vary between situations. That is, on one end of the spectrum, we will choose to implement just a few tests that only cover the code paths that we believe are most likely to contain a bug. On the other end of the spectrum, we will choose to implement a large suite of unit tests that are incredibly thorough and test a wide variety of scenarios. Wherever a given scenario falls on that spectrum, we will ensure to write unit tests that verify our program behaviors as well as what we are expecting in a normal sense and in more unexpected scenarios, like boundary conditions or error conditions. The following figure 20 shows what we will test when doing code unit tests structure and refactoring.



```perl
###############################################################
########################################################
#####################Extracting the ESSID
# Subroutines for ESSID
########################################################
####################
$data = $1 if $output =~ s/(^ath0[\s\S]+)ath0/ath0/sm;
sub getESSID {

       $data='';
       if ($essiddata =~ m/ESSID:"(\w+)"/gm){
                $data = $1;
                return $data;}
        else{
                print "Not possible\n";}
        }
my $Node_ESSID = &getESSID;

########################################################
#################### #Extracting the mode
# Subroutines for getMode
########################################################
sub getMode {
       $data = '';
       if ($mode =~ m/Mode:(\w+\-+\w+)/gm){
                $data = $1;
                 return $data;}
        else{
                print "Not possible\n";}
        }
my $Node_Mode = &getMode;
```

Figure 24 Unit testing Overview

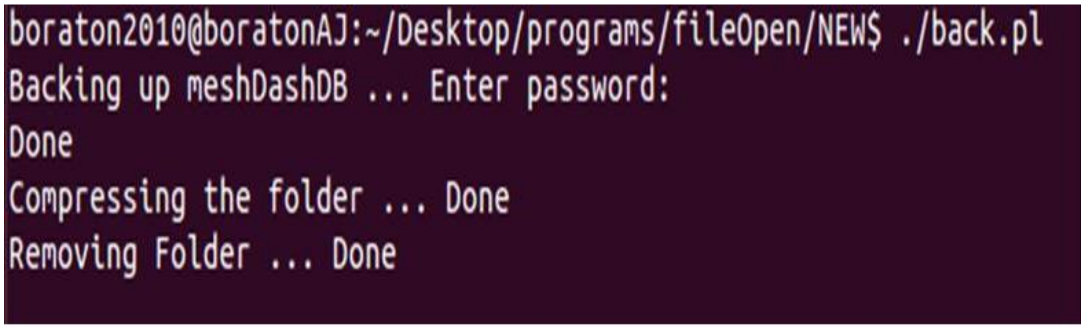
Our code refactoring as shown in the above figure, the unit testing will allow less code and improve the performance of the scripting application program for the backend monitoring. Therefore, performing unit testing on Perl program (such as performing benchmarking on our Perl application with benchmark module) is extremely fast when it comes to handling regular expression and test processing on both Mesh Potatoes and monitoring server.

Security Testing

We conducted security testing to protect against vulnerabilities. We implemented two security elements. The two included are:

1. Authentication : Credential verification between clients and servers
2. Authorization : By supplying user ID and password

This section of testing verifies that protection mechanisms built into a system will, in fact, protect this system from improper penetration [20]. Thus, the following diagram shows an instance of the security protection when taking the application database backup.



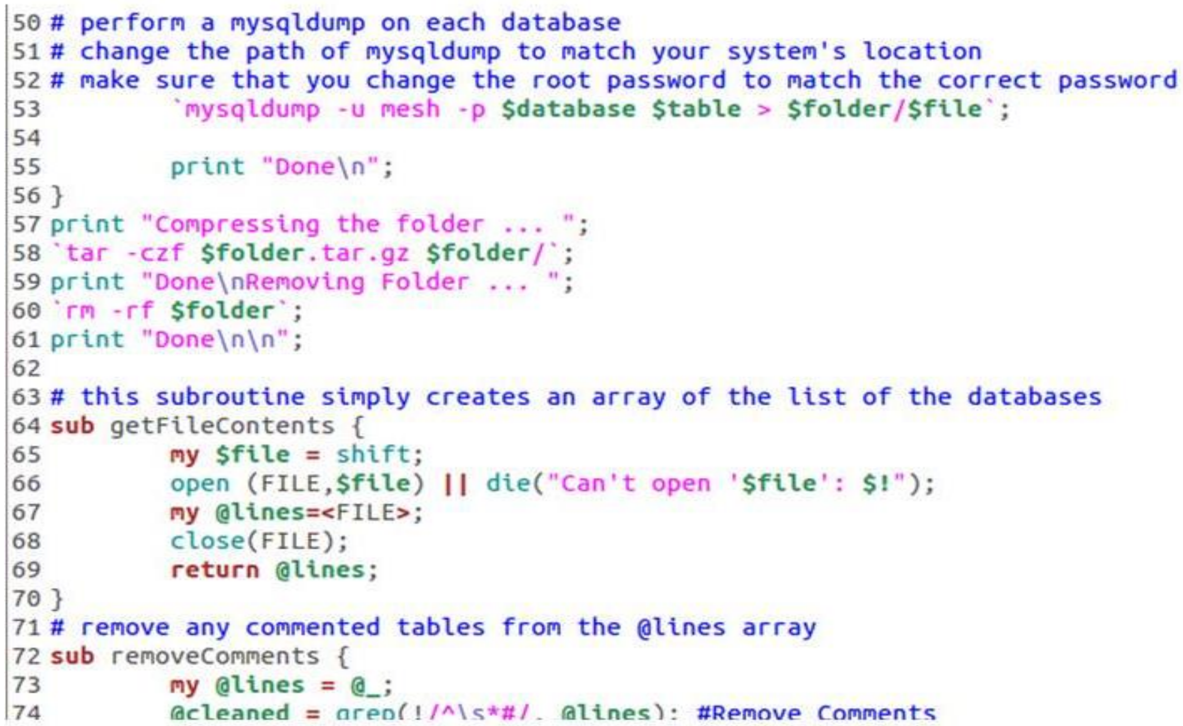
```
boraton2010@boratonAJ:~/Desktop/programs/fileOpen/NEW$ ./back.pl
Backing up meshDashDB ... Enter password:
Done
Compressing the folder ... Done
Removing Folder ... Done
```

Figure 25 Unit testing Overview

Furthermore, passwording the system using authentication and authorization method of security will avoid an outsider from penetrating into the system to causes system chaos.

Recovery Testing

This is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed [20]. This is critical to the system recovery. We conducted recovery testing by creating a backup of the entire database of the application. This is done automatic using a cron job for effective correctness and better system optimization. Thus, the following diagram shows the code overview of the backup and recovery code testing application.



```
# perform a mysqldump on each database
# change the path of mysqldump to match your system's location
# make sure that you change the root password to match the correct password
        `mysqldump -u mesh -p $database $table > $folder/$file`;

        print "Done\n";
}
print "Compressing the folder ... ";
`tar -czf $folder.tar.gz $folder/`;
print "Done\nRemoving Folder ... ";
`rm -rf $folder`;
print "Done\n\n";

# this subroutine simply creates an array of the list of the databases
sub getFileContents {
        my $file = shift;
        open (FILE,$file) || die("Can't open '$file': $!");
        my @lines=<FILE>;
        close(FILE);
        return @lines;
}
# remove any commented tables from the @lines array
sub removeComments {
        my @lines = @_;
        @cleaned = grep(!/^\s*#/, @lines); #Remove Comments
```

Figure 26 Backup Code Overview

Testing Issues and Results

Our testing strategy reviewed some issues which is most applicable to any network scripting application. These include:

1. Mesh Potatoes software versions: Checking that all nodes run the same batman-adv version. This is to avoid problems us with an incompatible versions of batman-adv running on Mesh Potatoes. Thus, incompatible nodes will simply ignore each other.
2. Precision and inaccuracy data: Accuracy refers to the closeness of a measured value to a standard or known value whereas precision refers to the closeness of two or more measurements to each other. If data are not captured correctly with the right regular expression syntax then this will affects the unit testing stage. Data must be capture rightly and test correctly to avoid mesh network and system failure.
3. Mixed mode operation: If operation of Mesh Potatoes interferes with other wireless around the site, then there will be network interference. It is important to take note of this in case of future network issues. Thus, such operation will affect our program

getting the right information to present to the front end application for the network manager virtualisation.

4. Incorrect arithmetic precedence and initialization: Wrong data calculation can result to test failure and also affect the application logic and database.

Chapter 9

CODE DOCUMENTATION & USER INSTALLATION GUIDES

Introduction

This chapter refers the intendant to the source code for implementing the backend application. Each source code is referred to as in appendices. Please see the appendix for more information about this project code documentation.

User Guides and Installation

This document section focuses on the installation of mesh application system for the backend monitoring for mesh network. The document covers the installation and basic configuration of mesh application system. Mesh application system is an open source implementation of a network activity for data and information capturing. Mesh application system is a LAMP (Linux + Apache + MySQL + PHP) application that interfaces with Perl DBI module using both the database driver (DBD) and open database connectivity (ODBC). This documentation has been tested using Debian operating system (Ubuntu 10.04 and Ubuntu 13.10).

Installations Steps

In a nutshell installing Mesh Application requires a minimum of eight steps (1-8):

1. Mesh Application System Installation Guides
2. Pre-required software packages
3. Installing the MySQL from package
4. Installing the phpMyAdmin from package
5. Installing the Perl, DBD::MySQL from package
6. A Simple Mesh Potatoes and Cron Jobs Set Up
7. Mesh Application Setup (Server)
8. Mesh Application Database Setup

Mesh Application System Installation Guides

The document describes or gives instructions for setting up the installation and configuration of DBD::MySQL, the Perl DBI driver for the MySQL database. The network or system administration will ensure that the require prerequisites are available (i.e. Perl, MySQL and DBI). This document also highlights the steps to take to install the LAMP for both the application for collecting the log file from Mesh Potatoes and backend application itself on the server [22].

Important note about distributions

This documentation assumes that the network/system administrator is using a .deb based distribution that has used the folder /usr/share/ during packaging. Other distributions use the alternate folder /var/lib/. The basic assumption of this documentation is that used pre-packaged software such as apache2 default root folder is /var/www. Likewise, phpMyAdmin which is a very popular MySQL management software packages will manage the data collection. Its installation will enable the network administrator to manage the data collection properly.

Pre-required software packages

Mesh application requires the packages of a LAMP (MySQL) and Perl DBI installation. To install the necessary packages, the network/system administrator must run the following commands on the terminal:

1. apt-get install php5-cli php5-mysql mysql-server apache2 php5-gd
2. apt-get install libapache2-mod-php5 php5 php5-common

LAMP applications are Wiki's, Content Management Systems, and Management Software such as phpMyAdmin. The traditional way the network/system administrator can install the LAMP applications is to:

1. Download an archive containing the application source files.
2. Unpack the archive, usually in a directory accessible to a web server.
3. Depending on where the source was extracted, configure a web browser to serve the files.
4. Configure the application to connect to the database.

5. Run a script, or browse to a page of the application, to install the database needed by the application.
6. Once the steps above, or similar steps, are completed the network/system administrator is ready to begin using the application for the mesh application.

Installing the MySQL from package

MySQL is a fast, multi-threaded, multi-user, and robust SQL database server. It is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software.

Installation

To install MySQL, the network/system administrator can run the following command from a terminal prompt:

1. sudo apt-get install mysql-server
   During the installation process the network/system administrator will be prompted to enter a password for the MySQL root user. Once the installation is complete, the MySQL server should be started automatically.
2. The following command can be run from a terminal prompt to check whether the MySQL server is running: sudo netstat -tap | grep mysql
3. When you run this command, you should see the following line or something similar: tcp 0 0 localhost: mysql *:* LISTEN 2556/mysqld
4. If the server is not running correctly, you can type the following command to start it: sudo /etc/init.d/mysql restart

Installing the phpMyAdmin from package

PhpMyAdmin is a LAMP application specifically written for administering MySQL servers [23]. Thus, this helps the network/system administrator to administer the mesh application for the backend network monitoring. Before installing phpMyAdmin the network/system administrator will need access to a MySQL database either on the same host as that phpMyAdmin is installed on, or on a host accessible over the network. For more information see *MySQL installation guide as stated previously*.

Installation

From the terminal prompt the network/system administrator can enter:

```
sudo apt-get install phpmyadmin
```

At the prompt the network/system administrator can choose which web server to be configured for phpMyAdmin. Once selected the rest of this installation will use Apache2 for the web server for complete installation.

To verify the Installation

In a web browser the network/system administrator can go to *http://servername/phpmyadmin*, replacing *serveranme* with the server's actual hostname. At the login page enter *root* for the *username*, or another MySQL user if you any setup, and enter the MySQL user's password. Once logged in you can reset the *root* password if needed, create users, create/destroy databases and tables, etc. However, in our case the mesh application on the server together with the Perl MySQL::DBD will create the database and tables automatically and also populate the tables with right data collection information.

Installing the Perl, DBD::MySQL from package

This is the most essential part of the documentation which describes the installation and configuration of DBD::MySQL. The Perl DBI driver for the MySQL database integration actually enables the mesh application to work properly. Without this the mesh application will not work. Thus, it is essential for the network/system administrator to take a proper precautionary of this installation guide [25].

Installation

The network/system administrator needs to install and configure Perl DBD::MySQL. Please note that DBD::MySQL is a DBI driver, hence network/system administrator need DBI for this mesh application to work. For a manual installation the network/system administrator need to fetch the DBD::MySQL source distribution. Please download the latest version. This can be found on [24]. The following are steps to follow to download and install the package:

1. Download from [24]. The name is typically something like:
   DBD-mysql-1.2216.tar.gz
2. From the terminal extracted the archive using the following commands : gzip -cd DBD-mysql-1.2216.tar.gz | tar xf –
3. This will create a subdirectory DBD-mysql-1.2216. Enter this subdirectory and type:
   perl Makefile.PL
   make
   make test
4. If the tests seem to look fine, the network/system administrator may continue with
make install

Configuration

The install script ``Makefile.PL" can be configured via a lot of switches. All switches can be used on the command line [25]. For our case we test the database using few codes from the mesh application to connect to MySQL database. The following command can be used on a terminal: perl Makefile.PL --testdb=<db>

A Simple Mesh Potatoes and Cron Jobs Set Up

In this simple mesh network the network/system administrator will set up a network of two or more MP devices so that phone calls or wireless connection can be made between them, and then connect one MP to a Local Area Network with Internet access so that a laptop can connect wirelessly to the virtual Access Point and access the LAN and Internet [7]. To set up an advance mesh network we can further referred to the user guides of the Mesh Potatoes to enable better device configurations.

In addition to these, on the Mesh Potatoes we setup the cron jobs program with a scripting program that call both static and dynamic shell program for collecting the mesh activity logs. The cron jobs are schedulers that automate the data collection in an effective and efficient way. Particularly, it enables the network/system administrator to execute the script program with timing.

The following are steps to take to set up the cron jobs on the MP. Please note that the path to the cron jobs is the same with the shell script (i.e. both static.sh and Dynamic.sh is schedule to automate data collection for varies MP commands) programs on the MP directory.

Installation

From the MP terminal: we configure recurring services. That is, recurring services are handled via the /etc/crontab. The network/system administrator can add the following cron jobs to the /etc/crontab or create a file with the jobs in /root/etc/crontab/Dynamic_Values.sh and /root/etc/crontab/Static_Values.sh on MP devices. The following scripts where schedule with the cron job on the MP:

1. Static_Values.sh: This script resides on MP and it will take care of the recurring service by using MP commands that are static in values to capture data which does not varies in values. The following is a cron job set up for the static values:

   75 * * * * /usr/bin/perl /root/etc/cron/ Static_Values.sh.

   This cron job keep log of the static values every 7 hours on the 5th month of the year. The network manager/system administrator can edit it to suit his data collections requirement.
2. Dynamic_Values.sh: This script resides on MP and it will take care of the recurring service by using MP commands that are dynamic in values to capture data which varies in values. The following is a cron job set up for the dynamic values:

   */15 * * * * /usr/bin/perl /root/etc/cron/Dynamic_Values.sh

   This cron job keep log of the static values every 15 minutes on the everyday in a year. The network manager/system administrator can edit it to suit his data collections requirement.

Mesh Application Setup (Server)

The network/system administrator can place the application to any path on the target system (i.e. server machine itself) and must set all the paths within the program code. A program path where the program itself can be executed. You can refer this to the program source code. The following diagram shows the complete integration of the application program into a one main program. The program can be found on my website for further instruction on how to run the script with the device.

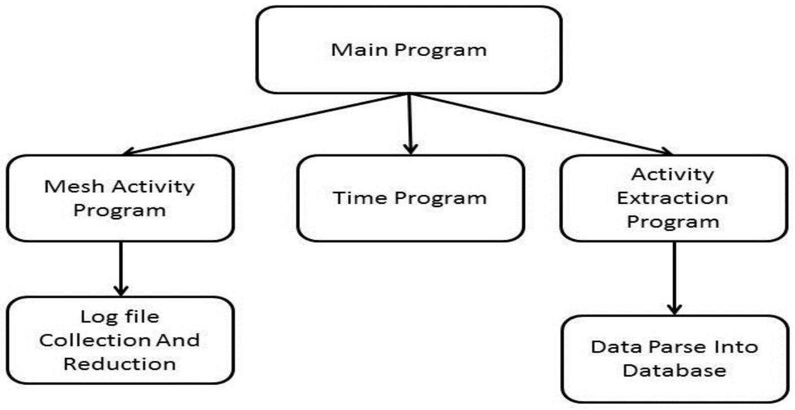


Figure 27: Main Integration Setup

Main Program

This program is setup to run all other programs since every other script depends on it. You can further refer to the source code of the mesh application. See Appendix B,C,D,E,F,G and also you can refer to the project website for the actual source code.

Mesh Activity program

This can be setup on the target system (server). See the website for further instruction

Back Up Program

Also see the website for instructions and program source code.

Mesh Application Database Setup

This was discussed earlier in this documentation. A network manager will need to setup MySQL database server to run mesh application on the server. The network manager can follows the subsequence instruction given within this documentation. Please refer to internet for further assistance or helps on how to install MySQL database.

Chapter 10

FUTURE RECOMMENDATION AND PROJECT CONCLUSIONS

This chapter concluded the entire project and briefly explains the future recommendation in very simple nutshell.

Future Recommendation

During the testing of the application, lecturers in the department of computer science (i.e. University of the Western Cape and external tester) suggested that instead of collecting and transferring the log files locally (i.e. manually) from the Mesh Potatoes, it is important to automate the script application, and have it logged into the device automatically. By so doing this, the device can exchange keys (such as public and private keys) for easy communication and also allow remote connection to either the server or the Mesh Potatoes. Therefore, we suggested that this fact can be a further research to this entire project.

Conclusion

In conclusion this project walked through software development life cycle, which is from project planning to project production and delivery. We describe each phase of the project by using agile software methodology of software development life cycle. We also carried out the experimentation setup to arrive at this project code and documentation. Particularly, during the testing phase we used different testing strategies to test our software functionality as well as its performance. Furthermore, we discussed test results for various components of the application and gave graphical and tabular representation of some of the components. The testing phase of this project was successful as all the testing strategies conducted were employed and successful. We also took a look into the results, and we analysis the results. Thus, there are certain project constraints that we encountered. We were recommended to researched and address those issues in a simplify ways and project specific domain. An efficient ways of debugging the entire system was possibly recommended for our future research.

Therefore, with this application the network/ system administrator can however be using the system.

APPENDIX A

Project Plan and Timeline
The following is the project plan for the backend application.

Terms	**Task Descriptions**
Term 1 Project Analysis User Requirements Document and Requirements Analysis Document	Understanding the problem facing Organization e.g. UWC Interview with Mr. Carlos Rey-Moreno to gather the user requirements. Meeting with the supervisor. Reading the specifications of the Mesh Potato and SECN firmware Installation of MySQL database, and Java database driver Still are leaning towards MadWifi and Batman-adv routing protocol.
Term 2 Designing and preparing the prototype. (Completed)	Documentation(Report) System Architecture User interface specification, Creating Mesh Network Setting-up Mesh Nodes, Designing and Configuration the batman-adv routing protocol on MP
Term 3 Implementation and coding	Further research to re-modify Carlos Script for project Implementation. To log real-time network activity. Capturing Signal received from neighbours. Working with the department Mesh Node for easy Integration. E.g. VOIP and Internet Usage. Re-modified Carlos Script for project implementation to log real-time Network Activity. SSH, SCP, Capturing, and Storing Mesh Activity received from mesh neighbours. Working with the department Mesh Node for easy Integration. E.g. VOIP and Internet Usage.

Term 4 Testing and Evaluation	Ability to test the application Remotely and locally with the help of Network Manager (e.g. Carlos and others). Test will be conduct through test bed. Qualitative evaluating Method of testing the App. That is, method of testing with some calculations and graphs representation. Setting up a life demo to demonstrate the application for real time network monitoring. Aim at testing with department mesh nodes. Application testing using the different methods of software engineering testing strategies. Analysis of the result will be gathered and explained. Presentation of the testing results and giving the feedback of the network manager Preparation of Report Document and Application User Guides

APPENDIX B

Perl Program: ETL activity program for the Mesh Potatoes

This section shows the Perl program using the concept of Extraction, Transformation, and Loading methods to get data into the MySQL database. A technique we used to arrive at the final project implementation. The source code is on my project website and you can download or view it. The following link we lead you to the website:

http://www.cs.uwc.ac.za/~oajayi/Downloads/Loading.pl

APPENDIX C

Mesh Potatoes: A shell program on MP for collecting a varies value collection

This section shows a dynamic shell program for capturing network activity and also helps to store the log file in Mesh Potatoes root directory.

```
#!/bin/bash
writeToResultFile() {
    touch log_`hostname`_$(date +%m%d%y).log
    ping 10.130.1.20 -c 20 >> /root/log_`hostname`_$(date +%m%d%y).log
    ping 10.130.1.21 -c 20 >> /root/log_`hostname`_$(date +%m%d%y).log
    batctl o >> /root/log_`hostname`_$(date +%m%d%y).log
    wlanconfig ath0 list >> /root/log_`hostname`_$(date +%m%d%y).log
    uptime >> /root/log_`hostname`_$(date +%m%d%y).log
    athstats >> /root/log_`hostname`_$(date +%m%d%y).log
    athstats >> /root/log_`hostname`_$(date +%m%d%y).log
    cat /proc/net/madwifi/ath0/rate_info >> /root/log_`hostname`_$(date +%m%d%y).log
    msg=" $*"
    echo "messaage: $msg"
    echo $msg >> log_`hostname`_$(date +%m%d%y).log
}
doProcess()
{
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
}
doProcess
```

APPENDIX D

Mesh Potatoes: A shell program on MP for collecting static data log

The following program shows a static shell program for capturing network activity and also helps to store the log file in Mesh Potatoes root directory.

```
#!/bin/bash
writeToResultFile() {
    touch log_`hostname`_$(date +%m%d%y).log
    batctl vm >> /root/log_`hostname`_$(date +%m%d%y).log
    batctl it >> /root/log_`hostname`_$(date +%m%d%y).log
    sysctl -a >> /root/log_`hostname`_$(date +%m%d%y).log
    iwconfig >> /root/log_`hostname`_$(date +%m%d%y).log
    ifconfig >> /root/log_`hostname`_$(date +%m%d%y).log
    cat /etc/bat-hosts >> /root/log_`hostname`_$(date +%m%d%y).log
    cat /etc/config/wireless >> /root/log_`hostname`_$(date +%m%d%y).log
    msg=" $*"
    echo "messaage: $msg"
    echo $msg >> log_`hostname`_$(date +%m%d%y).log
}
doProcess()
{
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
  writeToResultFile "ifconfig >> log_`hostname`_$(date +%m%d%y).log"
}
doProcess
```

APPENDIX E

Cron Jobs setup for both Static and Dynamic data collection

This section shows a cron scheduler for timing when to capture the network activity using the dynamic program and static program.

Dynamic Cron Job

```
75 * * * * /usr/bin/perl
/home/boraton2003/Desktop/Project/programs/fileOpen/scheduler/Dynamic_Values.sh
```

Static Cron Job

```
*/15 * * * * /usr/bin/perl
/home/boraton2003/Desktop/Project/programs/fileOpen/scheduler/Dynamic_Values.sh
```

APPENDIX F

A program for getting data collection out of Mesh Potatoes

This section of the appendix shows a Perl program for reducing the large log file kept in the Mesh Potatoes to a minimum log file. The following show how we implement the program. Please refer to link below for source code

http://www.cs.uwc.ac.za/~oajayi/Downloads/MeshActivity.pl

APPENDIX G

Performance Testing Program

```
#!/usr/bin/perl -w
use strict;
use warnings;
undef $/;
my $output = "time.text";
# Using the open() function to open the log output.
# Remove the newline from the log outputname
chomp $output;
unless(open (OUTPUT, $output)){
# Die with error message
# if we can't open it.
die "\nUnable to find the $output and fail for some reason: $!\n";exit;
  }
my $Num = 10;
print "\n";
print "\n";
print ">>>>>>>>>>>>>>  PERFORMANCE TESTING RESULTS
<<<<<<<<<<<<<<<<<<\n";
print "\n";
my $data='';
sub getSysTime {
      my $sysTime = shift; my $sumSys; my$aveSys;
      if ($sysTime =~ m/System time \(seconds\):\s+(\d+\.\d+)/s){
$data = $1;
my @sys = $sysTime =~ m/System time \(seconds\):\s+(\d+\.\d+)/gs;
my $arraysize = @sys;# Array size
foreach (@sys){
$sumSys+=$_;
$aveSys = $sumSys/$arraysize ;
```

```
print "Cumulative sum for the System time (seconds) is = $sumSys secs.\n";
print "Average per instructions for the System time (seconds) is = $aveSys sec.\n";
print "\n";
return @sys;}
else{
print "Not possible\n";}
}
sub getUserTime {
       my $Usertime = shift; my $sumUser; my $aveUser;
       if ($Usertime =~ m/User time \(seconds\):\s+(\d+\.\d+)/s){
$data = $1;
my @User = $Usertime =~ m/User time \(seconds\):\s+(\d+\.\d+)/gs;
#print "XXXXXUser>>>@User<<<XXXX\n";
my $arraysize = @User;
foreach (@User){
$sumUser+=$_;
$aveUser = $sumUser/$arraysize ;=
print "Cummulative sum for the User time (sceconds) is = $sumUser secs.\n";
print "Average per instructions for the User time (seconds) is = $aveUser sec.\n";
print "\n";
return @User;
}
else{
print "Not possible\n";}
}
sub getMax_Size {
      my $Max_Size = shift; my $sumMax; my $aveMax;
       if ($Max_Size =~ m/Maximum resident set size \(kbytes\):\s+(\d+)/s){
$data = $1;
my @Max = $Max_Size =~ m/Maximum resident set size \(kbytes\):\s+(\d+)/gs;
my $arraysize = @Max;
```

```
foreach (@Max){
$sumMax+=$_;
$aveMax = $sumMax/$arraysize ;
}
print "Total Maximum Paging Size for the User Application = $sumMax (bytes)\n";
print "Average per size is = $aveMax bytes\n";
print "\n";
return @Max;
}
else{
print "Not possible\n";}
}
sub getPercent {
    my $Percent = shift; my $sumPerc; my $avePerc;
     if ($Percent =~ m/Percent of CPU this job got:\s+(\d+%)\s/s){
$data = $1;
my @Per = $Percent =~ m/Percent of CPU this job got:\s+(\d+%)\s/gs;
my $arraysize = @Per;
my $l;my $CPI;
foreach $l (@Per){
$l =~ s/\%//g;
$sumPerc+=$l;
$CPI = ($sumPerc * $Num);
$avePerc = $CPI/$arraysize;
}
print "Total Maximum CPU clock cycles in seconds is = $sumPerc secs.\n";
print "Maximum Average CPU is = $CPI clock cycles per sec. \n";
print "Average Execution Time  = Average clock cycles per instructions is = $avePerc
Sec.\n";
print "\n";
return @Per;
```

```
}
else{
print "Not possible\n";}
}
while(my $output1 = <OUTPUT>){&getSysTime($output1 &getUserTime($output1);
&getPercent($output1);
&getMax_Size($output1);
}
close OUTPUT;
```

APPENDIX H

Back up Program for Database Recovery

This program will enable the network/ system administrator to recover database if peradventure the mesh application failed.

```
# We use this Perl Script to Backups the Mesh Application databases Which was specified
# in the dbbackup.config file
#!/usr/bin/perl
use File::Basename;

# We write the names of specified database into config file
# This program uses a comment to bypass any database that we don't want to backup
# That is, # Unwanted_DB (i.e. commented: will not be backed up)
# We set the directory to keep the backup files
# We make sure that the directory exists

$backup_folder = '/home/boraton2010/Desktop/programs/fileOpen/NEW'; #EDIT THIS LINE

# the config file is a text file with a list of the databases to backup
# this should be in the same location as this script, but you can modify this
# if you want to put the file somewhere else
my $config_file = dirname($0) . "/dbbackup.config";

# retrieve a list of the databases from the config file
my @databases = removeComments(getFileContents($config_file));

# change to the directory of the backup files.
chdir($backup_folder) or die("Cannot go to folder '$backup_folder'");

# grab the local time variables
my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
$year += 1900;
$mon++;
#Zero padding
$mday = '0'.$mday if ($mday<10);
$mon = '0'.$mon if ($mon<10);
# create the name of the backup folder that will contain all of the backup files
my $folder = "$year-$mon-$mday";
mkdir($folder) or die("Cannot create a folder called '$folder'");
# backup each database contained in the @databases array
```

```
foreach my $database (@databases) {
            next if ($database eq '');
            chomp($database);
            my $table = '';
            if(index($database,' ')+1) { #Get just 1 table in the database - if there is a ' '(space) in the db name
                        my @parts = split(' ',$database);
                        $database = $parts[0];
                        $table = $parts[1];
            }
# you may comment out this print statement if you don't want to see this information
print "Backing up $database ... ";
            my $file = $database;
            $file .= '_' . $table if($table ne '');
            $file .= ".sql";
# perform a mysqldump on each database
# change the path of mysqldump to match your system's location
# make sure that you change the root password to match the correct password
            `mysqldump -u mesh -p $database $table > $folder/$file`;
            print "Done\n";
}
print "Compressing the folder ... ";
`tar -czf $folder.tar.gz $folder/`;
print "Done\nRemoving Folder ... ";
`rm -rf $folder`;
print "Done\n\n";
# this subroutine simply creates an array of the list of the databases
sub getFileContents {
            my $file = shift;
            open (FILE,$file) || die("Can't open '$file': $!");
            my @lines=<FILE>;
            close(FILE);
            return @lines;
}
# remove any commented tables from the @lines array
sub removeComments {
            my @lines = @_;
            @cleaned = grep(!/^\s*#/, @lines); #Remove Comments
            @cleaned = grep(!/^\s*$/, @cleaned); #Remove Empty lines
return @cleaned;
}
```

APPENDIX I

Aging Script

This scripting will assist the network/system administrator to age the data logged on the Mesh Potatoes after collecting relevant information from the generated log file on the Mesh Potatoes.

```
#!/usr/local/bin/perl
use strict;
my $TODAY = time;
my $DIR = '.';
#my $LOGFILE = "/var/spool/uv/test/fileout";
my $wktime = 604800; (ie 86400*7)
##################################################################################
sub write_to_file{
    open (LOGFILE, ">/var/spool/uv/test/fileout");
    opendir DIR, $DIR or die "could not open directory: $!";
    while (my $file = readdir DIR)
    { next if -d "$DIR/$file";
    my $mtime = (stat "$DIR/$file")[9];
  if ($TODAY - $wktime > $mtime) {
    print LOGFILE "$DIR/$file is older than 7 days...removing\n";
         }
    }
}close LOGFILE; close DIR;
####################################################################################
sub remove_old_files{
    opendir DIR, $DIR or die "could not open directory: $!";
    while (my $file = readdir DIR){
    next if -d "$DIR/$file";
    my $mtime = (stat "$DIR/$file")[9];
    if ($TODAY - $wktime > $mtime){
    unlink $file;}
    }
}close LOGFILE; close DIR;
#######################################################################################
Main:{
    write_to_file();
    remove_old_files();}
```

REFERENCES

[1] Aichele, C., Wunderlich, S., Lindner, M., and Neumann, A. *Better Approach to Mobile Ad-hoc Networking (B.A.T.M.A.N.)* draft-wunder lich-openmesh-manet-routing-00. 2008.

[2] Self-study: *Broadening the concept of b.a.t.m.a.n-adv routing protocols. (2006)*. Retrieved March 03, 2013, from open mesh website: http://www.open-mesh.org/projects/batman-adv/wiki

[3] Crow, B.P., Indra, W., Sakai, P.T., and Jeong Geun, K. IEEE 802.11 *Wireless Local Area Network. IEEE* Communications Magazine, September (1997), 116-126

[4] Marlo, J. (2006). *GPRS remote access and data collection for rural telehealth project (Honours thesis, University of the Western Cape).* Retrieved from http://www.uwc.ac.za/~mjooste

[5] M.E.M. Campista, P.M. Esposito, I.M. Moraes, L.H.M. Costa, O.C.M. Duarte, D.G. Passos, C.V.N. de Albuquerque, D.C.M. Saade, and M.G. Rubinstein, *"Routing Metrics and Protocols for Wireless Mesh Networks,"* IEEE Network, vol. 22, Jan. 2008, pp. 6-12.

[6] Self-study: *Understand Mesh Potato devices and firmware. (2007).* March 01, 2013, Retrieved from the Village telco website: http://villagetelco.org/mesh-potato/

[7] Self-Study: *Rendered the concept of a backend and frontend of a system. (2013)*. Retrieved March 25, 2013. *How to set up mesh network for backend* system: www.shutterstock.com

[8] Shepherd M. N. (2011). *An overview of Wireless Mesh Network Protocols and Voice over IP considerations (Honours thesis, Unversity of Cape Town).* Retrieved from http:www.uct.ac.za

[9] Self-study: *ReMesh, Mesh Network Workgroup*. May 12, 2013. Retrieved from http://mesh.ic.uff.br.

[10] J. Duarte, D. Passos, R. Valle, L. Magalhães, D. Saade, C. Albuquerque. *Management Issues on Wireless Mesh Networks, Latin-American Operation and Network Management Symposium* – LANOMS 2007, Petrópolis, RJ, Brazil. September 2007.

[11] R. De T. Valle, D. Passos, C. Albuquerque2, D. C. Muchaluat. *Mesh Topology Viewer (MTV): an SVG Based Interactive Mesh Network Topology Visualization Tool*. To be published in IEEE Network Magazine, 2007.

[12] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, R. Rosati. *A principled approach to data integration and reconciliation in data warehousing*. In Proc. Intl. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, (1999).

[13] H. Galhardas, D. Florescu, D. Shasha and E. Simon. Ajax: *An Extensible Data Cleaning Tool*. In Proc. ACM SIGMOD Intl. Conf. On the Management of Data, pp. 590, Dallas, Texas, (2000).

[14] M. Golfarelli, D. Maio, S. Rizzi. *The Dimensional Fact Model: a Conceptual Model for Data Warehouses.* Invited Paper, International Journal of Cooperative Information Systems, vol. 7, n. 2&3, 1998.

[15] M. Golfarelli, S. Rizzi: *Methodological Framework for Data Warehouse Design*. In ACM First International Workshop on Data Warehousing and OLAP (DOLAP '98), pp. 3-9, November 1998, Bethesda, Maryland, USA.

[16] R. Kimbal, L. Reeves, M. Ross, W. Thornthwaite. *The Data Warehouse Lifecycle Toolkit*: Expert Methods for Designing, Developing, and Deploying Data Warehouses. John Wiley & Sons, February 1998.

[17] Self-study: *Syntax (programming languages)*, August 23, 2013. Retrieved from Wikipedia website http://en.wikipedia.org/wiki/Syntax_ (programming_languages)

[18] Badonnel, R., State, R. and Festor, O. (2005), *Management of mobile ad hoc networks: information model and probe-based architecture*. Int. J. Network Mgmt.15: 335–347. doi: 10.1002/nem.577

[19] Batin, C., Seri, S., and Navate, S.B, (1994) *Conceptual Database Design: An Entity Relational Approach*, Redwood City, California

[20] Roger S. Pressman, Software Engineering: A Practitioner's Approach, Sixth Edition, McGraw-Hill Internation Edition 2005, pp 387-647.

[21] Self Study: http://mgarner.wordpress.com/2006/09/27/automated-testing-for-datamarts/, Accessed September 12th, 2013.

[22] Self Study: http://cpansearch.perl.org/src/JWIED/DBD-mysql-2.1028/INSTALL.html, Accessed September 12th, 2013.

[23] Self Study: https://help.ubuntu.com/community/phpMyAdmin, Accessed September 12th, 2013.

[24] Self Study: http://www.cpan.org/modules/by-module/DBD/, Accessed September 12th, 2013.

[25] Self Study: http://cpansearch.perl.org/src/JWIED/DBD-mysql-2.1028/INSTALL.html, Accessed September 12th, 2013.