Containerized IoT Solution for Efficient Mobile Vertical Handover

by

Amit Singh Gaur

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

Abstract

Internet of Things (IoT) is ubiquitous, which includes objects communicating through heterogenous networks. One of the challenges in mobile IoT is vertical handover decision (VHD) between heterogenous networks for seamless connectivity. Conventional VHD approach is based on received signal strength (RSS), which is limited to only RSS Quality and hence inefficient to decide best network for vertical handover. This thesis proposes multi-criteria based VHD (MCVHD) algorithm for efficient VHD between Wi-Fi, Radio and Satellite network. The experiment results show that MCVHD outperforms the conventional RSS Quality based VHD by minimizing handover failures, unnecessary handovers, handover time and cost of service by selecting best available network using multi-criteria parameters. The proposed IoT solution also employs Docker container based microservices architecture. The results show that the Docker container produces negligible resource overhead and can be used on resource constrained IoT devices like Raspberry Pi 3, for efficiently managing IoT application and services.

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisor Dr. Chung-Horng Lung for his immense support, guidance and motivation during my MASc thesis research.

I would also like to extend my thanks to Dr. Ioannis Lambadaris for his support and guidance which has helped me to accomplish my research.

Sincere thanks to Natural Sciences and Engineering Research Council of Canada (NSERC) and Ontario Centres of Excellence (OCE) for funding my thesis research.

Finally, I would like to take this opportunity to thank my parents for their enormous love and support. Their word of encouragement always keeps me motivated. Special thanks to Jyoti Budakoti for always being there.

Amit Singh Gaur

List of Abbreviations

AMQP	Advanced Messaging Queuing Protocol
API	Application Programming Interface
ARP	Address Resolution Protocol
CDB	Configuration Database
CLI	Command Line Interface
COAP	Constrained Application Protocol
CSV	Comma-Separated Values
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EPC	Electronic Product Codes
IEEE	IEEE Institute of Electrical and Electronics Engineers
IETF	IETF Internet Engineering Task Force
GPL	General Public License
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ICMP	Internet Control Message Protocol
ID	identifier
IoT	Internet of Things
IPV6	Internet Protocol Version6
IP	IP Internet Protocol

IT	Information Technology
JSON	JavaScript Object Notation
L3	L3 Layer 3
LAN	LAN Local Area Network
MAC	MAC Media Access Control
MCVHD	Multi-Criteria Vertical Handover Decision
MADM	Multi-Attribute Decision Making
MPLS	MPLS Multi-Protocol Label Switching
MQTT	Message Queue Telemetry Transport
MT	Mobile Terminal
M2M	Machine to Machine
NETCONF	NETCONF Network Configuration Protocol
NFC	Near Field Communication
NFV	Network Function Virtualization
NMS	Network Management System
NoSQL	Not Only Structured Query Language
OS	OS operating system
OUI	Organizationally Unique Identifier
PHP	PHP Hypertext Preprocessor
P2P	Point to Point
QoS	Quality of Service
REST	Representational State Transfer
RF	Radio Frequency

RFID	Radio Frequency Identification
RPC	RPC Remote Procedure Call
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	TCP Transmission Control Protocol
TOPSIS	Technique to Order Preference By Similarity To Ideal Solution
TTL	TTL Time to Live
UDP	UDP User Datagram Protocol
UAV	Unmanned Aerial Vehicle
URI	Uniform Resource Identifier
VHD	Vertical Handover Decision
VM	Virtual Machine
WLAN	wireless local area network
XML	Extensible Markup Language
UI	User Interface
WWW	World Wide Web

Table of Contents

List of Tables

List of Figures

Chapter 1 Introduction

Recently Internet of Things (IoT) has drawn a lot of attention and is envisioned in various sectors in the near future due to its promising benefits. A well-rounded definition of Internet of Things (IoT) by "IEEE IoT Initiative" refers to "Internet of Things envisions a self-configuring, adaptive, complex network that interconnects 'things' to the Internet using standard communication protocols. The things offer services, with or without human intervention, through the exploitation of unique identification, data capture and communication, and actuation capability. The service is exploited by intelligent interfaces and is made available anywhere, anytime, and for anything taking security into consideration" [1]. Figure 1.1 presents the pictorial representation of the IoT concept.

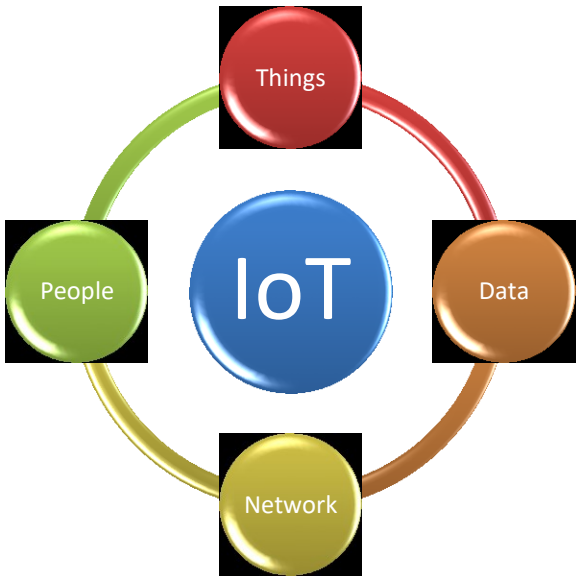

Figure 1.1 Internet of Things (IoT) Concept [1]

In the last few years, the IoT paradigm has created revolution in a number of sectors and greatly impacted our life in various possible ways. The increasing application areas has made the growth of the IoT devices taking an exponential curve as shown in Figure 1.2 [2].

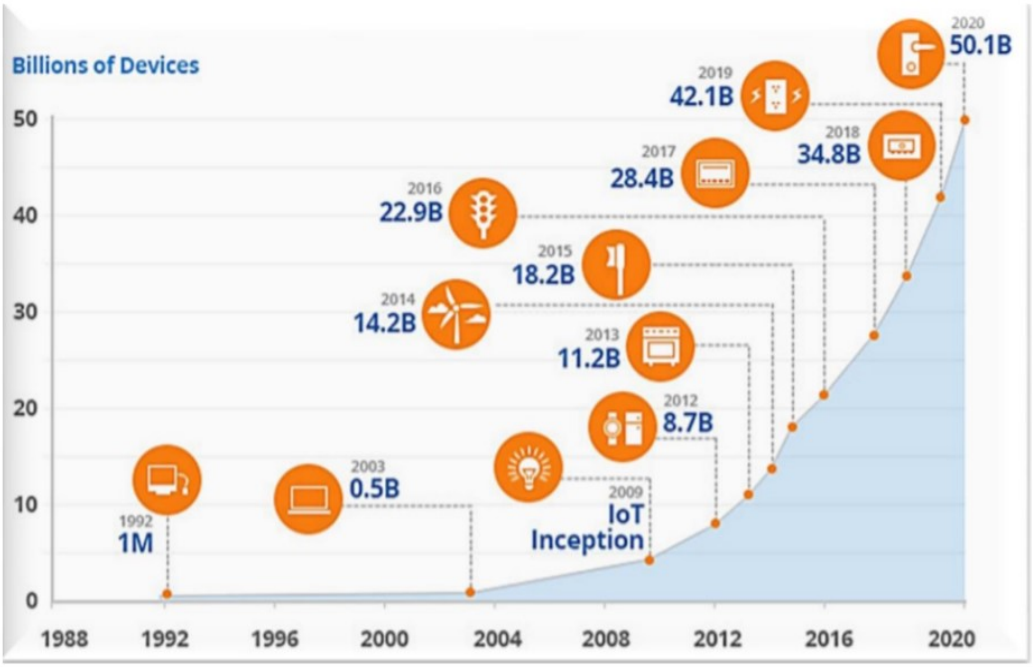

Figure 1.2 Rise in connected devices showing exponential curve [2]

As the applications continue to grow, it is changing the nature of the devices that are being attached. This opens a path of IoT devices to be the natural means of communication, control, and development. The ability of these systems to connect and share useful information via the Internet is becoming ubiquitous. In many cases, enormous amount of data is generated from embedded sensors that need to be processed in an efficient way along with the required computation power.

One of the highly popular emerging use cases of IoT is IoT-equipped Mobile Terminals (MTs). The MTs can range from vehicles on road to ships on water to UAVs/drones in the air. Sensors attached to these MTs can stream data for tracking and monitoring to cloud applications using IoT services. As the MT traverses, it switches among different available networks to satisfy its needs in terms of quality of service. This process is referred to as Vertical Handover due to a transition among different heterogenous networks and the decision to select best available network is referred to as Vertical Handover Decision (VHD). For IoT network disruptive services, the MTs need to effectively handle vertical handover as the MT move from one access area or cell to another. This requires an efficient VHD mechanism for delivering IoT services [3]. A vertical handover takes place among the access points supporting different network technologies. Figure 1.3 shows a scenario of vertical handover when a mobile node is moving from cell A of Wi-Fi network to cell B of Radio network.

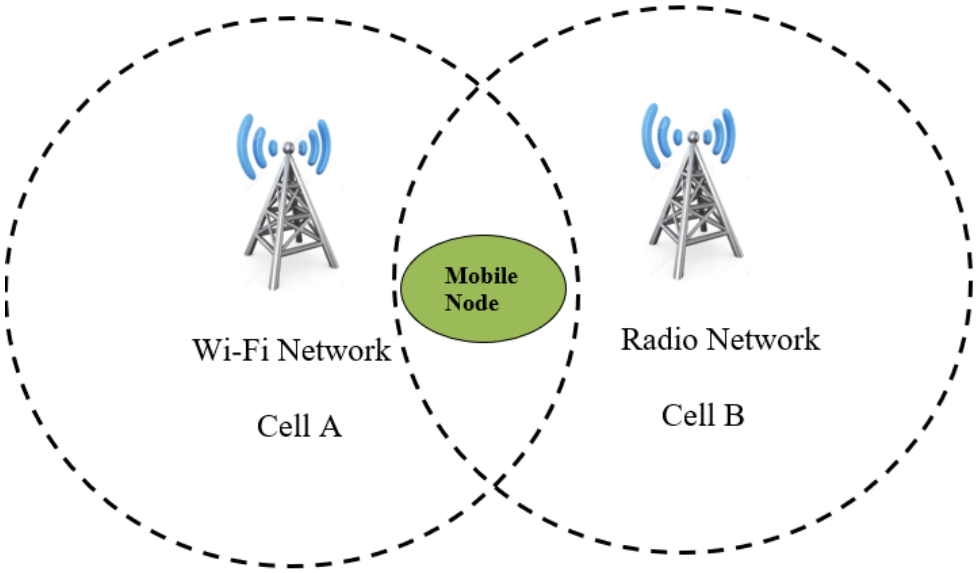

Figure 1.3 Mobile node moving from Cell A to Cell B [3]

1.1 Problem Statement

Vertical handover process is a power and resource consuming task which imposes challenges for efficient delivery of IoT services. The first major challenge is that typically IoT devices are resource constrained, which means they may have extremely reduced capabilities. An efficient IoT service requires a light weight application which requires low processing, storage and power overhead. The second major challenge is that IoT devices stream enormous amount of data and the selected network for data transmission could impact the cost associated with the provided network services. For example, Satellite network services are very costly compared to Wi-Fi network services. Hence, selection of an efficient network is important for data transmissions in IoT. Selection of a network among multiple available networks using different technologies could depend on different parameters of priority but the application must choose one that has low cost to transfer massive volume of data. The third challenge is to provide seamless connectivity to Mobile IoT devices, also referred to as Mobile Terminal (MT) in this thesis, for IoT applications which require asset tracking.

This thesis resolves these challenges by presenting an efficient and light weight multi-criteria based containerized solution for IoT services to handle efficient vertical handovers between W-Fi, Radio and Satellite networks. The proposed multi-criteria based vertical handover decision (MCVHD) algorithm makes a tradeoff between various parameters like received signal strength (RSS), quality, latency, cost, network range, data rate, mobility, etc. Recently, the container technology has become very popular due to its lightweight architecture, hence the thesis presents a containerized solution which provides easy deployment of the services and user specified remote change in service.

1.2 Research Objectives

To overcome the potential problem in tracking and monitoring mobile nodes seamlessly within line of sight and beyond line of sight for wireless IoT services, the main objective of this research is to build an efficient vertical handover algorithm for IoT applications using the concept of container based virtualization.

The proposed MCVHD algorithm based on multi-criteria parameters continuously measures multiple attributes like RSS, data rate, latency, network coverage range, associated link cost and other parameters for all the available communication interfaces. Then, based on priority decision matrix, the proposed algorithm seamlessly performs vertical handover in case of a network failover. In addition, the proposed approach makes use of the container based virtualization technique. Container based virtualization means that the proposed solution uses OS-level virtualization method and can be deployed and launched as a standalone isolated application solution. The benefit of using the container based architecture is to reduce processing overhead for power and resource constrained IoT devices by deploying solution in isolated lightweight containerized environment.

The proposed algorithm makes use of the ping protocol and RS232 to communicate to the attached serial Modem device and obtain interfaces connectivity information and monitor the device by sending ICMP packets and then the MCVHD algorithm decides which network to handover in case of a network failure. This IoT failover solution has been developed using the Docker container [4] to provide performance efficiency in terms of availability, security, and isolation.

1.3 Contributions

The main contributions of this research thesis are summarized below:

1) The main scientific contribution is the proposed MCVHD algorithm for efficient IoT services in heterogeneous networks.
2) This thesis also explores the potential benefits of containerized vertical handover solution which is system independent and thereby adaptable and easily deployable. The solution uses Docker containers which makes services rapid deployable, portable and efficient version control for advancement in the services.
3) The thesis also explores the use of network function rate limiter for limiting data rates in all three networks, i.e., Wi-Fi, Radio and Satellite to reduce data transfer cost as per network selection.
4) The thesis also evaluates the performance of the container technology versus the native environment for running storage and computing-intensive IoT applications on resource constrained IoT Gateway device on the edge device of a network.

1.4 Thesis Outline

This thesis is organized as follows:

Chapter 2 discusses relevant background information about vertical handover and vertical handover decision methods and summarizes the results of the literature study.

Chapter 3 describes the design and implementation of the proposed multi-criteria based vertical handover decision (MCVHD) algorithm with detailed information on design architecture and data processing.

Chapter 4 presents the implementation of containerized IoT solution for vertical handover mechanism.

Chapter 5 discusses about the experimental setup environment and performance evaluation of the proposed IoT solution.

Chapter 6 concludes the thesis, discusses the goals which have been reached and suggests the future works.

Chapter 2 Background and Related Work

This chapter explores the background information and related work in IoT and vertical handover challenges in mobile IoT. This chapter also focuses on the handover mechanism using multi-criteria for vertical handover decision (VHD) making. Section 2.1 introduces IoT, its elements, protocols and architecture; it summarizes the related technologies in IoT networks. One of the major challenges faced by the mobile IoT is handover. Section 2.2 presents introduction to Handover and discusses about the VHD criteria, classification of VHD algorithms and performance evaluation metrics for vertical handover algorithms. Section 2.3 describes virtualization techniques and management systems as a lightweight solution to resource constrained devices. Section 2.4 discusses the literature review of vertical handover methods in comparison to the proposed solution in the thesis. The summary of this chapter is discussed in Section 2.5

2.1 Internet of Things

The rapid growth of wireless technologies and the Internet have made IoT an integral part of the scientific research. The idea behind this concept is to make communications even more pervasive and agile without requiring any human interactions. A detailed overview of IoT elements and standards is presented in following sub-sections.

2.1.1 IoT Elements

Typically, there are five elements which define the functionality of IoT [5], as shown in

Figure 2.1. Each element is further elaborated as follows.

Figure 2.1 Elements of IoT [5]

1) **Identify:** Identification is essential to IoT for identifying, addressing and matching IoT devices and services. The most common methods for identification are electronic product codes (EPC) and ubiquitous codes (uCode) [6]. IPv6 and IPv4 are two common IP addressing methods. 6LoWPAN is also widely used as discussed in papers [7]
2) **Sense:** Sensing in IoT refers to collecting information from different IoT devices over a network. The collected information is then send to the data storage repositories for further analysis and computing.
3) **Communicate:** Various communication protocols like Wi-Fi, ZigBee, Bluetooth, IEEE 802.15.4, RFID, Z-wave, and LTE-Advanced etc. are used for communication in IoT. Some other communication technologies like Near Field Communication (NFC) and ultra-wide bandwidth (UWB) are also popular in IoT communication.
4) **Services and Analytics:** IoT services can be categorized into three classes. The first, Identity-related services are used identify the physical and virtual objects. The second, Information Aggregation Services is used to collect the sensor data and aggregate it to be used by other IoT application services. The third, Collaborative-Aware Services,

take the data from the Information Aggregation Services and use the processed data to make the further decisions.

5) **Compute:** The IoT platforms provide the facilities for IoT devices to send their data to local storage, cloud or for big data to be processed in real-time. These compute nodes provide deep analytics and help users to analyze data and take appropriate decisions. For hosting IoT services there are many free and commercial cloud platforms available by different organizations.

Table 2.1 summarizes common operating systems used in IoT environment.

Table 2.1 Common Operating Systems used IoT Environment [8]

Operating System	Language Support	Minimum Memory (KB)	Event-based Programming	Multi-threading	Dynamic Memory
TinyOS	nesC	1	Yes	Partial	Yes
Contiki	C	2	Yes	Yes	Yes
LiteOS	C	4	Yes	Yes	Yes
Riot OS	C/C++	1.5	No	Yes	Yes
Android	Java	_	Yes	Yes	Yes

2.1.2 IoT Common Standards

World Wide Web Consortium (W3C), Internet Engineering Task 17 Force (IETF), EPCglobal, Institute of Electrical and Electronics Engineers (IEEE) and the European Telecommunications Standards Institute are some of the groups responsible to provide IoT protocols and standardization for the programmers and the service providers.

Figure 2.2, summarizes the important protocols defined by these groups and the following subsections provide a brief overview of IoT application protocols.

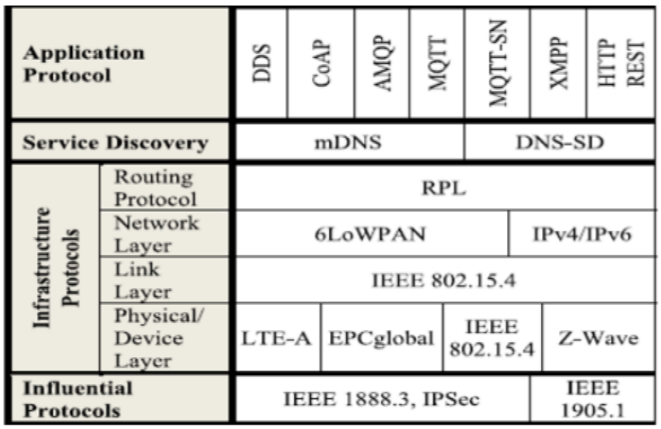

Figure 2.2 Standardization efforts in support of the IoT [8]

2.1.3 Application Protocols

1) **Constrained Application Protocol (CoAP):** CoAP is an application layer protocol created by IETF Constrained RESTful Environments (CoRE) for the IoT applications. The CoAP is based on REST (REpresentational State Transfer) which is used similar to as HTTP. REST is a stateless client server architecture. It has the same methods as used in HTTP i.e. post, get, put, and delete. REST enables the clients as well as the servers to expose and consume web services like the SOAP (Simple Object Access Protocol) but in an easier way using URIs (Uniform Resource Identifiers) [9]. The only difference is that CoAP is built over UDP instead of TCP and CoAP is lighter than HTTP which utilizes minimum power, computation and communication capabilities for REST interactions and which makes it more favorable for resource constrained IoT devices.

2) **Message Queue Telemetry Transport (MQTT):** MQTT is a based on publish subscribe architecture and provide support for Middleware and M2M communication. MQTT supports resource constrained devices by utilizing low bandwidth links[10]. MQTT is buid over TCP protocol and it delivers messages through three levels of the QoS. The publishers create a topic and send data as per the topic to the broker. The subscribers registered to the particular topic by the broker receives the published data.

3) **Extensible Messaging and Presence Protocol (XMPP):** XMPP allows instant messages to communicate on the Internet and help in achieving authentication and encryption. In XMPP protocol the gateways can act as bridge between the external messaging networks. XMPP runs over a wide variety of Internet based platforms in the decentralized fashion.

4) **Advanced Message Queuing Protocol (AMQP):** AMQP is an IoT application layer protocol which is built over TCP. AMQP also supports publish/subscribe model for communication. AMQP uses Exchanges to route the messages to the queues based on pre-defined rules [11]. AMQP Queue also buffers messages and later send them to the subscribers/receivers. Table 2.2 presents comparison between IoT protocols.

Table 2.2 Comparison between different IoT protocols [8]

Application Protocol	REST ful	Transport	Publish/ Subscribe	Request/ Response	Security	QoS	Header Size (Bytes)
COAP	✓	UDP	✓	✓	DTLS	✓	4
MQTT	×	TCP	✓	×	SSL	✓	2
MQTT-SN	×	TCP	✓	×	SSL	✓	2
XMPP	×	TCP	✓	✓	SSL	×	_
AMQP	×	TCP	✓	×	SSL	✓	8
DDS	×	TCP UDP	✓	×	SSL DTLS	✓	_
HTTP	✓	TCP	×	✓	SSL	×	_

2.1.4 NFV with IoT

NFV is an initiative driven by largest service providers to increase use of virtualization and the commercial servers in their networks. NFV leverages the IT technologies, including virtualization and open software to fundamentally change the way the networks are built, deployed and operated. Leading service providers have begun a significant network transformation led by the implementations of NFV platforms. NFV provide technology to customize the network to the IoT requirements.

NFV's ability to distribute the intelligence throughout network enables real-time analytics and business intelligence. NFV provides menu of the virtual applications that can be combined to deliver customized network services required by the IoT. Figure 2.3 shows relocation of NFV from servers to IoT Gateway.

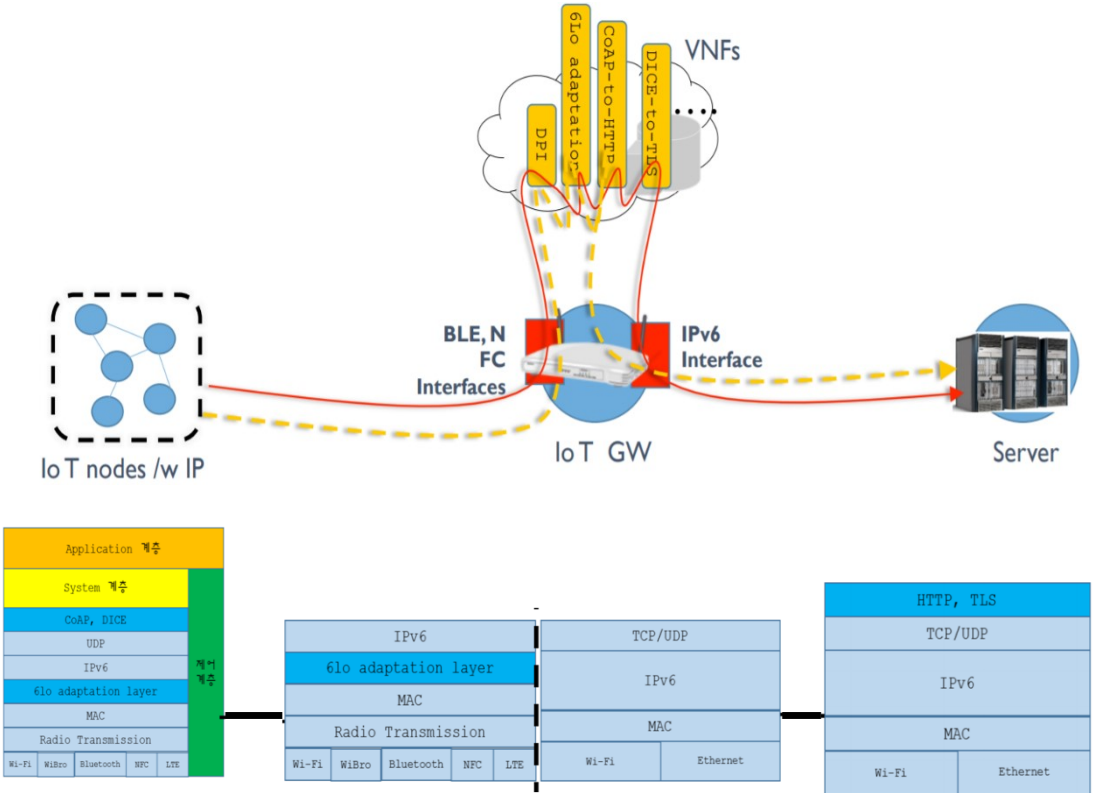

Figure 2.3 Relocating IOT Gateway Network Function [12]

2.2 Handover

The IoT application areas where IoT nodes are mobile faces challenges to support seamless mobility, hence mobility management becomes essential in mobile IoT. Figure 2.4 represents a transportation market incorporating multiple mobile IoT nodes (terminals)

Figure 2.4 Global IoT In Transportation Market [13]

One of the major challenges faced by mobility management in IoT is handover between heterogenous networks to support seamless connectivity as per QoS.

Handover is the process of transferring a user's active session from one network to other when mobile terminal moves away from the connected network. Network handovers is classified into two types, horizontal handover and vertical handover.

1) **Horizontal handover (HHD):** A horizontal handover takes place between two different access points of the same network, e.g. cellular network.
2) **Vertical handover (VHD):** a vertical handover takes place between different network technologies, e.g. Wi-Fi, Satellite network

An important part of handover process is the vertical handover decision (VHD) based on selecting best available network among all the candidate networks. In contrast to the RSS based handovers, the VHD algorithms consider other criteria as well like, cost of the service, network coverage, mobile terminal (MT) velocity and power consumption. Aligning with the research objectives of this thesis, the objective is to design an efficient multi-criteria based VHD algorithm to improve network selection and user satisfaction.

2.2.1 VHD criteria

As per the literature review below is the brief overview of some important criteria for VHD:

1) **Received signal strength (RSS)***:* RSS is highly popular criteria for VHD because of easy measurement and is directly related to the service quality of signal. RSS quality also depends on distance from mobile terminal (MT) to its point of attachment (POA) [14]. Various of existing horizontal handover algorithms have RSS as the main decision criteria for handover, and similarly, RSS is an important criterion for VHD algorithms as well.
2) **Network connection time***:* The network connection time is referred to the connection time provided by a network's point of attachment to the MT. The variation of the RSS due to distance and velocity of the MT from the network's point of attachment can affect the network connection time of the MT to a particular network. Heterogeneous

networks like Satellite network and Wi-Fi network etc. have different network coverage sizes hence for VHD the network connection time plays a significant role.

3) **Available bandwidth*:*** Available bandwidth is defined as measurement of data that can be transmitted during fixed given time and expressed in bits per seconds (bps). It is also an important criterion for VHD and especially become highly important for delay sensitive application.
4) **Power consumption***:* Power consumption is critical especially in resource constraint IoT MT, hence Vertical Handover to a network point of attachment is preferable which would support extending battery life of the MT.
5) **Cost of Network Service*:*** Cost of network service should be taken into consideration by VHD as different networks can have varied cost associated. For example, service cost of data transmission through Satellite network is higher than data transmission through Wi-Fi or Cellular network.
6) **Security***:* Security is also an important aspect for VHD in those applications where confidentiality of data is critical. For such applications a network with higher data security level is selected in handover decision.
7) **User priority:** A user preference can prioritize the selection of the favorable access network among other available heterogeneous networks.

2.2.2 Classification of VHD algorithms

VHD algorithms can be classified based on the handover decision criteria used. Below listed are four types of VHD algorithms:

1) **RSS based algorithms*:*** RSS is the main criteria in RSS based VHD algorithms and is conventional method in handover mechanism. Many methods have been proposed and

developed to select best network among available heterogeneous networks based on highest RSS value. Since the heterogenous networks show disparity of the technologies involved [14] with different thresholds which makes comparison difficult, hence, other network parameters like Quality factor or bandwidth are usually combined with RSS for handover decision.

2) **Bandwidth based algorithms*:*** Bandwidth is an important criteria in VHD specially in delay-sensitive applications. This category of VHD algorithms is based on available bandwidth for a MT [16]. Both bandwidth and RSS are also used in some VHD algorithms [17].
3) **Cost function based algorithms*:*** These VHD algorithms selects the best network based on comparing cost function. A cost function typically involves monetary cost, security, bandwidth and power consumption [18]. These metrics are evaluated on the basis of user preference weights and network conditions.
4) **Combination algorithms*:*** This category of VHD algorithm involves large number of parameters which makes it very difficult to evaluate for VHD. Hence, to solve this complexity these algorithms use machine learning techniques to formulate the VHD process [19] [20]. Artificial neural network is also used to create the VHD algorithms where comprehensive set of input-desired output patterns are available [20]. A real-time learning processes is applied to the system which can analyze and accordingly modify their own structure to create very effective VHD algorithms.

Table 2.3 summarizes a comparison evaluation of different VHD algorithms on the basis of criteria metric.

Table 2.3 Comparison of different VHD algorithms [15][16]

Group	Applicable networking technologies	Input parameters	Handover target selection criteria	Complexity	Reliability
RSS based VHD algorithms	Usually between macrocellular and microcellular networks	RSS as the main input	The network candidate with the most stable RSS	Simple	Reduced reliability because of the fluctuation of RSS
Bandwidth based VHD algorithms	Between any two heterogeneous networks	Bandwidth combined with other parameters such as RSS	The network candidate with the highest bandwidth	Simple	Reduced reliability because of the changing available bandwidth
Cost function based VHD algorithms	Between any two heterogeneous networks	Various parameters such as cost, bandwidth and security	The network candidate with the highest overall performance	Complex	Reduced reliability because of the difficulty in measuring some parameters
Combination algorithms	Between any two heterogeneous networks	Different input parameters depending on different methods	The network candidate with the highest overall performance	Very complex	High reliability because of the training of the system

2.2.3 Performance evaluation metrics for VHD algorithms

VHD algorithms can be quantitatively compared under various usage scenarios by measuring the metrics:

1) **Handover time:** The duration between initiation and completion time of the handover process is referred to as handover time. The initiation corresponds to the instant when the VHD algorithm selects the best network and successfully connects to it. The completion time refers to the time taken by the first acknowledgement packet to be

received by the MT after handover. Reducing delay in handover time is important for delay-sensitive applications.

2) **Number of handovers**: Reducing the number of unnecessary Vertical Handovers is usually preferred for constrained IoT devices as frequent unnecessary handovers would cause wastage of network resources, processing overhead, wasted of power consumptions, costs, etc. Ratio of unnecessary Vertical Handover can be defined as the ratio of the number of unnecessary Vertical Handover to the total number of Vertical Handover occurred during a period of time.
3) **Handover failure:** A Handover failure occurs when the Handover request is initiated but the handover is not completed successfully. There could be several reasons for Handover Failures such as unavailability of sufficient resources to complete the handover to the target network or MT moves away from the coverage range of the targeted network before completion of handover process.
4) **Throughput*:*** Throughput of network is defined as the rate of data transfer to the MT on connected network. Higher throughput is important for applications which are latency intolerant hence, it is desirable for VHD to select network with higher throughput.

The critical issues to be taken into consideration while developing IoT system with a handover mechanism is that typically IoT devices are resource constrained and only lightweight solutions should be adopted for efficiency. The other factor which affects the design of such a system is the interoperability with different sensors manufactured by different vendors. These issues can be resolved by using the efficient virtualization

technique enabling lightweight solution and easy application development and deployment. The next section discusses about the virtualization technology for developing efficient IoT solutions.

2.3 Virtualization Technology

Virtualization refers to the abstraction of hardware resources by utilizing software programmability. Virtual computing environment improves resource utilization by providing a unified integrated operating platform for users and applications based on aggregation of heterogeneous and autonomous resources [21]. In terms of Linux, virtualization creates multiple Linux operating systems on a single host computer, thus known as Linux virtualization. Virtualization commonly has two approaches; hypervisor-based and container-based virtualization, which are described in next section.

2.3.1 Hypervisor-Based Virtualization

Hypervisor is a virtualization technique, also known as Virtual Machine Monitor that creates and runs virtual machines (VM). A hypervisor operates on top of a single host computer and allows users to run several additional operating systems, which are referred to as guest operating systems.

Hypervisor allows physical host machine to run multiple virtual machine instances, each running an independent operating system and applications, by sharing the same pool of virtualized hardware resources. Hence providing efficient resource utilization and reduced hardware cost. There are two types of hypervisors [22] which are also illustrated in Figure 2.5.

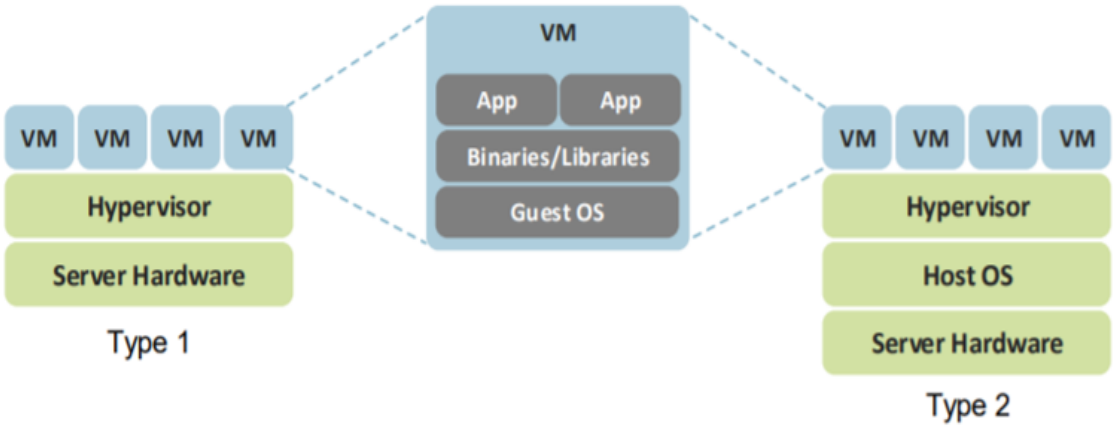

Figure 2.5 Type1 and Type 2 Hypervisor based virtualization architecture [22]

Type 1 hypervisors directly run on top of a system hardware and manage the guest operating systems with the help of hardware resources. In this type of hypervisors, the virtual machines are isolated from the hardware because they run on a separate level. While Type 2 hypervisors are termed as hosted hypervisors which run as a software layer inside the traditional host operating system.

2.3.2 Container – Based Virtualization

Container-based virtualization, is a lightweight virtualization technique that virtualizes the hosting operating system instead of the hosting hardware, hence also known as operating system-level virtualization. Container provides a different level of abstraction in which a kernel is shared between containers and more than one process can run inside each container.

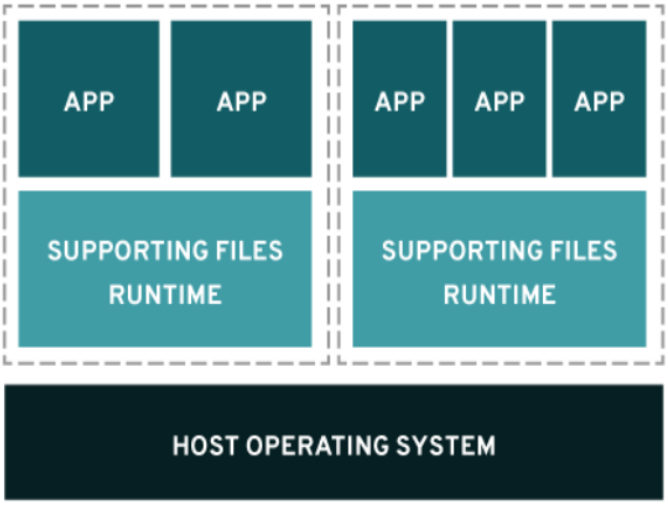

Figure 2.6 Container based virtualization runs applications on isolated users pace instances but on the kernel of the host OS [23]

Figure 2.6 shows two containers running on top of a host operating system utilizing system resources. The advantage of using a container-based virtualization is to create a number of virtual instances on a single host machine, as the kernel and other libraries are shared between those instances, thus uses less disk storage as compared to hypervisor-based solutions. Table 2.4 shows a comparison between Virtual Machines and Containers [24].

Table 2.4 A comparison between Virtual Machines and Containers [24]

Parameters	Virtual Machines	Containers
Operating System (OS)	Each virtual machine runs on top of a server hardware (Type-1 hypervisors) or a host OS (Type-2 hypervisors) and kernel is assigned individually to each VM with full hardware resources	Each container runs on top of a host operating system with kernel and other hardware resources shared between those guest operating systems

Startup time	It takes few minutes to boot virtual machine	Containers can boot up in few seconds depending upon the system specifications
Storage	Hypervisor-based virtualization takes much more storage because whole system components have to be installed and run including kernel	The storage is less as compared to VM because OS is shared and so is the kernel
Communication	Typically, network devices are used for communication if VMs run on different servers but in case of same server, standard IPC mechanisms such as signals, sockets, pipes, etc. are used	Same is the case for containers but standard IPC mechanisms are more generally used
Performance	There is an additional layer of hypervisor software which degrades performance as it takes time to translate machine instructions from guest OS to host OS	There is no additional layer as container provides near native performance since it is running on top of a host

2.3.2.1 Introduction to Linux Containers

LXC is an operating system level virtualization method for running multiple isolated linux systems on a controlled host using a linux kernel. The Advantages of Linux containers [25] as are follow:

1) **Portability**: Linux containers can run in any environment without changing the functionality of the operating system. The applications running inside a container can also be bundled together and then deployed onto various environments.
2) **Fast application delivery:** Containers are fast to build because they are considered as a light-weight virtualization and it takes seconds to build a new container vs minutes

for a VM. System developers and administrators can easily deploy applications into the production environment, as the work-flow of containers is easy to interpret. In addition, they provide good visibility to the developers and reduces the time for development and deployment.

3) **Scalability:** It is easy to run and deploy containers in any Linux system, cloud platform, data-centers, desktop computers and many other environments. Moreover, containers can also be scale up and scale down quite rapidly meaning that you can scale up containers from one to one thousand and then scale them down again. Thus, Linux containers are suitable for scale out applications that are deployed in cloud platform.
4) **Higher density workloads:** With Linux containers, one can run huge amount of applications on a single host system. Since containers do not use the full operating system, the resources are efficiently utilized between different applications as compared to hypervisors.

As containers have less overhead compared to full virtual machines, it is easier to have many isolated applications running with less resources, which is always a factor worth considering in IoT systems. Nowadays, a software called Docker [4] has gained a prominent position in the field of container-based virtualization technologies. Docker is a tool that facilitates the creation and management of application-specific containers by providing a toolset and an API layer for utilizing underlying kernel-level technologies, such as LXC containers, cgroups and overlay filesystems. Together, these technologies provide containers, which are isolated from each other in terms of filesystem, resources and network stack while allowing the containers to share a common base for their respective filesystems, thus saving disk space.

2.3.2.2 Introduction to Docker Containers

Docker is an open platform to build, deploy and run distributed applications using Docker engine and Docker hub [44]. Docker engine is a portable, lightweight runtime and packaging tool while Docker Hub is a cloud service for sharing applications and automating workflows.

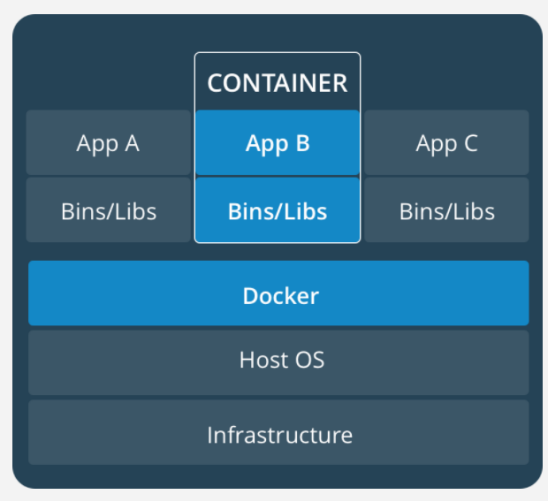

Figure 2.7 Docker Container [4]

Docker containers isolate the applications which make containerized applications easy to deploy on any machine running on docker environment. The isolation of the application provides independent development, testing and management of each application. Thus, multiple containers can run simultaneously on a single host without interfering with each other.

Apart from local Docker instance, there are several IaaS providers that are supporting running version of Docker containers. Some of the examples are OpenStack , Google Cloud Platform , Microsoft Azure, Amazon EC2, and Carina.

2.3.2.3 Docker architecture and components

Docker is based on a client-server architecture. The overview of Docker architecture is shown in Figure 2.8. It is composed of the following core components [4]:

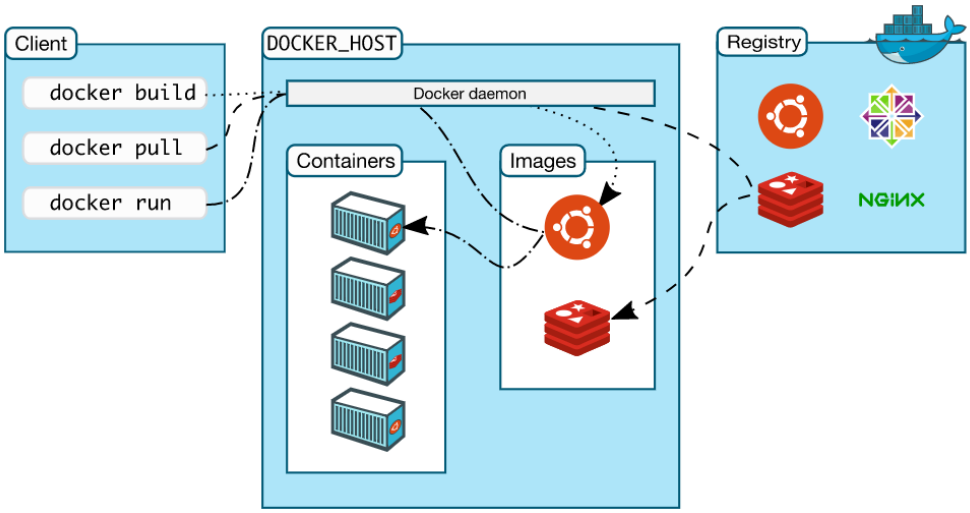

Figure 2.8 Architectural diagram of docker [4]

The main components of Docker architecture are as follows:

1) **Docker daemon:** The main functionality of Docker daemon (*dockerd*) is to listen to Docker client API calls and manages Docker images and Docker containers. It builds, runs and distributes Docker containers.
2) **Docker client**: Docker client (*docker*) helps in interacting with the Docker daemon. It sends docker commands to the Docker daemon using Docker API.
3) **Docker images:** Docker images are main component of docker architecture as it contains the instructions for creating the Docker containers. Images can be build by using the instructions set and template provided by Docker. Docker images are built with the help of *Dockerfile.* Dockerfile contains the set of syntax for each step needed for creating Docker images. By running a dockerfile, each instruction runs and creates

an intermediate layer in the image and after completing all instructions final image is formed. It is also possible to modify the existing image or reuse the already available image by downloading from Docker public repository. Modifying the dockerfiles and rebuilding the images from dockerfile do not create all layers instead rebuild and creates only changed layers. Hence, this property of docker makes images small, lightweight and fast in comparison to other virtualization technologies. Therefore, containerized solutions are portable, easy to modify/create, portable and easily.

4) **Docker registries**: Docker registries are used to store the Docker images. DockerHub is a public repository provided by Docker. It is cloud based storage for uploading new created images or downloading the already available images in the repository. There are many opensource Docker images available such as Ubuntu OS image, web-service image, database server image etc. Docker hub also provide private cloud storage where user can store the own private Docker images. Private cloud storage is secured and accessibly only to authorized user.
5) **Containers:** Docker images are used to build the Docker containers. Docker containers contain applications with required dependencies, environment and libraries to run in a loosely isolated environment. Running container contains one or more related process attached to the application which it contains. Containers can be easily start, stop, delete and move with help of Docker commands.

2.4 Related Work

Due to rapid increase in IoT devices and internet technologies, the realization of internet of everything is not far behind. In parallel, the increase and the fast evolution of several

IoT platforms which can support multiple interfaces, allowing IoT devices to roam seamlessly and connect with any technologies such as Wi-Fi, 3G, Radio, WiMAX, ZigBee and Satellite, anytime and anywhere. However, providing seamless connectivity to IoT devices in heterogeneous networks (Wi-Fi, Cellular, Radio, WiMAX and Satellite) is challenging task. So, role of effective vertical handover management in heterogenous wireless networks has gain significant popularity.

Various vertical handover algorithms have been proposed in research literature work. Research work [26] presents a framework for comparing different vertical handover algorithms based on Quality of Service (QoS). Though the framework shows impressive results, but the research work is limited only to two vertical handover algorithms. Research work [27] presents survey related to design and performance issues with comparison of several vertical handover algorithms but the research work is limited to only RSS Quality based function.

Research work [28], [29] based on RSS criteria to make VHD. Network which has the highest RSS value is selected among available networks. Similarly research work in [30],[31] discusses about various handover methods but has the same drawback of only considering RSS value. The RSS value can fluctuate more often hence can result in unnecessary handovers or probably handover failures [32]. Thus, this approach which is based only on RSS value is not efficient to handle vertical handover in heterogenous networks. Another research work [33] presents a handover algorithm but it only considers fixed bandwidth as a decision criterion and probably would fail during dynamic bandwidth decision scenario. Some other research works [34] considers other criteria like SINR, mobile terminal velocity etc. Though the approach to involve multiple criteria is

appreciable but the research work is much concentrated to a specific scenario which lacks flexibility.

Other research work has included other multiple criteria for handover decision like cost function based, network context aware, multi attributes decision matrix (MADM), artificial intelligence etc. cost function based algorithms [35], [36] tries to minimize the cost function of metrics such as bandwidth, delay, jitter, security, etc. But there are certain limitations using these methods. The cost based method is good in user- centric based network selection but, not necessarily prove to be good in satisfying user satisfaction level.

The handover decision based on multiple-attributed decision making (MADM) method is more popular as it considers multi-criteria for selecting best available network. Similar approach like MADM method is presented in research work [37], which shows comparison of TOPSIS methods for VHD. It considers criteria like RSS, RTT, bandwidth, latency, jitter, range, etc. This method first eliminates the networks which do not meet the requirement and then heavy computation to give each network a score to decide the best network. Though this approach is better than the rest of the other VHD methods but it lacks in terms of more use of resources like CPU, memory, power due to heavy computation. Hence, this kind of approach is not suitable for the resource constrained IoT devices. Another VHD method is proposed in research work [38] which is based on WPM This method first calculates weights of each criterion and then generates decision matrix. Based on network parameter values. After calculating the cross product of the weights and the decision matrix the result is normalized. Finally the best available network is chosen. Though this approach is good, but the proposed scheme is imbalanced with respect to the QoS of a network. One of the MADM based method, called TOPSIS decision model is

most popular VHD method due to its simple calculations, less computation and higher efficiency. Research work [38] is based on the TOPSIS technique. Research work [39] has enhanced the TOPSIS technique by combining two approaches i.e. multiple-AHP and E-TOPSIS. The M-AHP has been used to assign weights to network criteria while E-TOPSIS has been used for ranking network.

Some research work based on artificial intelligence algorithms [40], works on fuzzy logic, genetic algorithms and neural networks techniques to interpret imprecise information, and convert imprecise data into crisp numbers. Research work to [41][42][43], focuses on utility functions models, are commonly used to describe the level of satisfaction of the mobile terminal according different services offered by the network technology.

The literature consists of various schemes based on the methods stated above, but unfortunately none of the schemes provides vertical handover in a generic way. Thus, using one scheme can be better in one particular scenario but it may be not good in another scenario. We reviewed most of these schemes from different perspectives, and we found that these schemes had several issues such as using imbalance parameters for network selection, inappropriate handover triggering, complex computation during handover, wrong network selection, etc.

Based on this literature review, the MADM methods are largely used to optimize the vertical handover problem. Our objective in this thesis is to put forward a hybrid approach based on the TOPSIS algorithm which is one of MADM methods and the artificial algorithms based on fuzzy logic. The TOPSIS algorithm is used to calculate the performance concerning each candidate network. Fuzzy logic is applied to convert

imprecise data into numbers based on conversion scales and the best network based on higher ranking calculated by TOPSIS is selected for handover.

In the literature review there was not much research work available for a container based multicriteria vertical handover algorithm that considers multiple attributes decision making with MADM methods for optimal network selection considering Wi-Fi, Radio and Satellite as available heterogenous networks. However, there are only limited studies partially addressing this issue. The thesis explores the potential of containerized solution for vertical handovers in chapter 4.

2.5 Chapter Summary

This Chapter presents introduction about IoT concept and components of IoT. IoT has spurred manufacturing of IoT devices, and delivery of applications services in every domain. One of the most challenging domain is Mobile IoT. Mobile IoT struggles with the network connectivity issues and hence a need for efficient handover mechanism.

Handover is a process of transferring user connection from one access point to other. If the handover takes place between same network, its referred as horizontal handover while if handover takes place between different networks ten it is referred to as vertical handover. Many research works have been reviewed to study vertical handover decision (VHD) making methods to select best available networks. Most of the VHD methods are found to be either RSS based, or Cost based. RSS based VHD are inefficient as it only considers RSS value to determine best network, however due to fluctuating nature of some network this method can lead to unnecessary handovers. The cost based method measures either delay, service cost etc. parameters to determine cost function. An efficient VHD

method was found to be TOPSIS technique, which is based on multiple-attributed decision making (MADM) method. This method considers multi criteria to determine best network for handover, however it fails in certain conditions where network parameters are imprecisely determined

Another area of the literature review is about the virtualization technology. Nowadays Docker container technology is widely popular and hence this thesis reviewed the docker technology in terms of application over resource constrained IoT devices.

Chapter 3 Design and Implementation of the Proposed Approach

Different technologies along with emerging paradigm has been described so far with its various application areas. This chapter discusses the key terminologies and introduces the conceptual design and implementation of the proposed IoT solution.

This chapter is comprised of three sections. Section 3.1 presents the system architecture and hardware components used in the proposed IoT solution. The major hardware components used are sensor nodes, IoT Gateway, Radio Modem, Satellite Modem and Wi-Fi. IoT Gateway collects the data from sensors and does data processing and transmit data through either Wi-Fi, Radio or Satellite link depending upon best available network as presented in Section 3.2. Section 3.3 describes conventional RSS Quality based VHD algorithm and introduces the proposed MCVHD algorithm based on multi criteria. Section 3.4 presents an algorithm to set the rate of data transmission for Wi-Fi, Radio and Satellite network. The summary of the chapter is described in Section 3.5.

3.1 Design Overview and Configuration of IoT System

The key parts of this thesis are the Sensing Subsystem, Processing Subsystem, Communication Subsystem and Power Subsystem. Sensing Subsystem is comprised of JY901 sensor nodes which has accelerometer, magnetometer and gyroscope sensors. Processing Subsystem comprised of Gateway node designed using Raspberry Pi 3 and Communication Subsystem is comprised of Wi-Fi, Radio and Satellite Modem for streaming IoT data. Power Subsystem provides battery power to all Subsystems.

The Sensor node streams sensor data to Raspberry Pi 3 which acts as a IoT Gateway node with a baud rate of 9600 bps. The Raspberry Pi 3 collects, filters and combines sensor data from multiple streams and then transmits data to the base station server using Wi-Fi or Radio or Satellite communications. Figure 3.1 shows the design of the IoT system and Figure 3.2 depicts the main Subsystems.

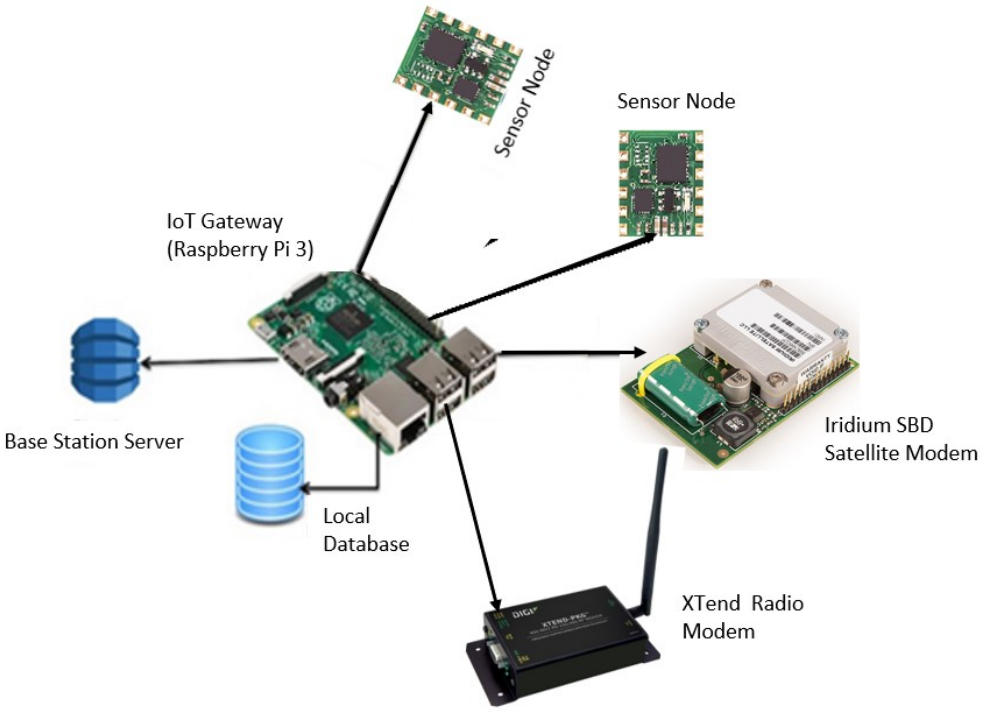

Figure 3.1 Design of the proposed IoT system

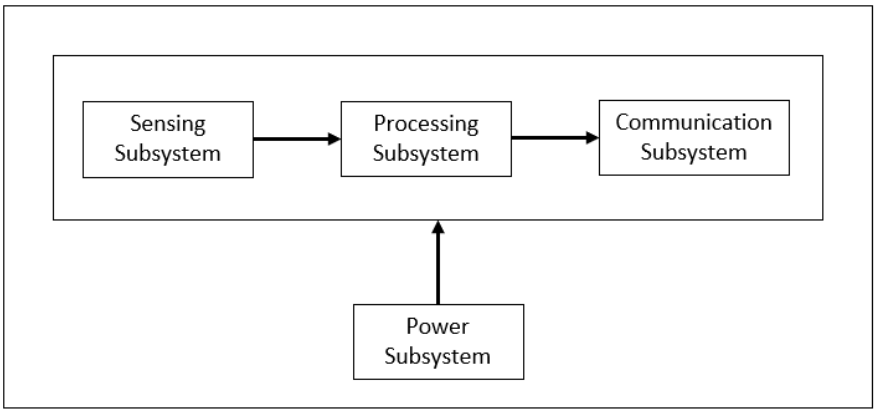

Figure 3.2 System architecture based on subsystems

3.1.1 Sensing Subsystem

This subsystem consists of JY901 sensor node which integrates high precision gyroscope, accelerometer and geomagnetic sensor. The sensing subsystem sends the data collected from different sensors to the processing subsystem.

3.1.1.1 Sensor Node

In the proposed design, we have chosen to use JY901 sensor node which has accelerometer, magnetometer and gyroscope with extended functionality using GPS. This type of sensor is suitable to track the movements of the objects. JY901 sensor specifications [45] are summarized as below:

1) Gyroscope in JY901 measures angle, acceleration and angular velocity
2) It has a 3-axis gyro sensor, a 3-axis accelerometer sensor, and a 3-axis magnetic sensor.
3) It uses Kalman filter combining with the gyro and accelerometer to get a high precision of the angle measurement.
4) It can be used for various applications, such as 4-axis flight control and self-balancing robot and the module can get accurate attitude in a dynamic environment with a precision of 0.05 degrees.
5) It uses a high-precision gyro accelerometer MPU6050, which reads the measurement data by a serial port and an IIC port. The sensor uses advanced digital filtering technology with minimum noise to improve the accuracy.

3.1.2 Processing Subsystem

The processing subsystem is used to process the data collected by sensing subsystem. The processing subsystem consists of a microprocessor with storage. This subsystem is the brain of the proposed IoT system. Its main tasks are to: (i) aggregate and process the data obtained from the sensing subsystem, process, (ii) runs MCVHD algorithm for vertical handover and finally (iii) run rate limiter algorithm and forward the processed data to the communication subsystem for data transmission to the base station server. This subsystem also has storage to store the data locally.

3.1.2.1 IoT Gateway Data processing node

In the proposed design, the Raspberry Pi 3 [46] is used as the IoT Gateway Data processing node. The Raspberry Pi 3 is single board computer developed by the Raspberry foundation in UK and is a low cost, compact device with a rich set of open-source tools. The Raspberry Pi 3 microprocessor can do more complex and complicated tasks as it is built on BCM2835 system on chip (SoC), which is a highly efficient ARM1176JZF-S Cortex- A7 based quadcore processor that runs at 900MHz [46]. It has 1GB of RAM and has four USB ports. It has a built-in Wi-Fi module for data communications. It has a slot for MicoSD card which can be used to increase the storage up to 32 GB. Raspberry Pi 3 has embedded Linux-based operating system, Raspbian which can be connected remotely through SSH.

The IoT Gateway node receives the sensor data coming from multiple sensors and stores the data in the SQLite Database [47], which is a very light server-less database and does not burden the constrained device like Raspberry Pi 3 for processing.

3.1.3 Communication Subsystem

The role of the communication subsystem is to provide communication capabilities to the IoT system to transmit sensor data to the base station. This system consists of various mode of communication such as Wi-Fi, Radio Modem and Satellite Modem. Radio and Satellite Modem are serially connected to USB ports of Raspberry Pi 3 (IoT Gateway). Figure 3.3 shows the proposed IoT system with Wi-Fi, Radio and Satellite network used to transmit sensor data to base station server.

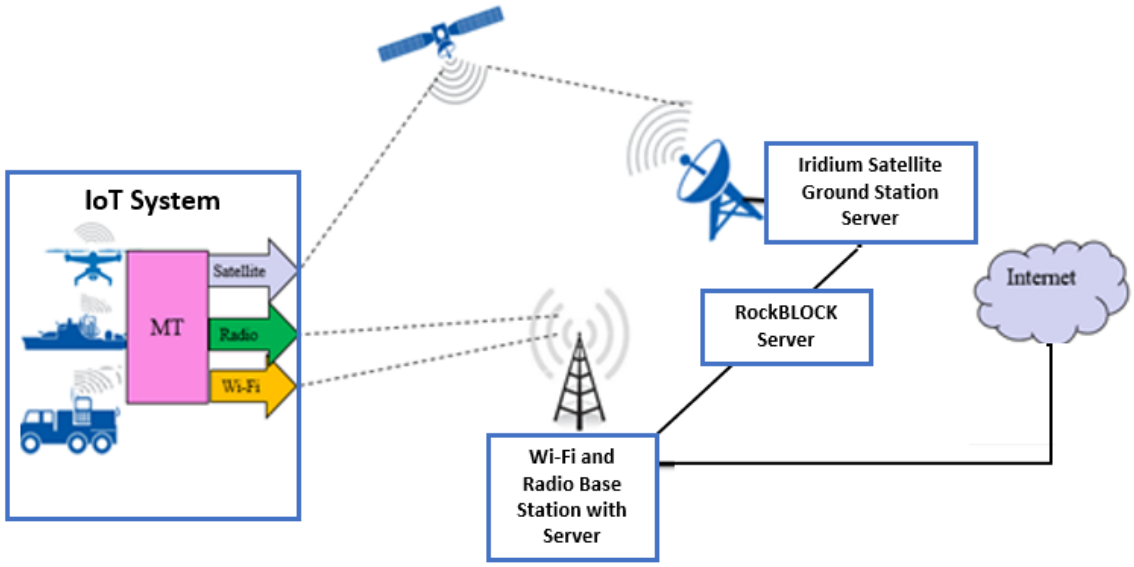

Figure 3.3 Communication in heterogenous network architecture

The mobile terminal (MT) could be any vehicle like drones, ships, lorries, etc., where the designed portable IoT system could be placed. The IoT system has Raspberry Pi 3 with built-in Wi-Fi, 9600 MHz Radio Modem and RockBLOCK Mk2 9602 Iridium SBD Modem for communications on the Wi-Fi link, Radio link, Satellite links, respectively.

A. Wi-Fi for Data Communication

Wi-Fi is a local area wireless networking technology which connects electronic devices on 2.4 GHz or 5 GHz Radio bands. The WLAN protocol is based on IEEE 802.11 standards

and has a range of 30 – 50 meters and high bit rate with bandwidth from 54 Mbps to maximum 600Mbps [48]. For secure connection, Wi-Fi supports encrypted technologies like WEP, WPA, WPA2, etc. Nowadays most devices have a built-in Wi-Fi module and connect to the Internet via a WLAN network. The Wi-Fi module for this thesis is built-in in the Raspberry Pi 3.

B. Radio Modem for Data Communication

Radio Modems can transfer data wirelessly across a range of up to tens of kilometers. Private Radio networks are used in critical industrial applications, when real-time data communications are needed. Radio Modems enable user to be independent of telecommunication or satellite network operators. The licensed frequencies are used either in the UHF or VHF bands. Using licensed frequency can ensure less likelihood of any radio interference from other RF transmitters. However, there are more chances of interference using license free frequencies because its free frequencies and other users can also use, hence chances of frequency blocking and interference increases.

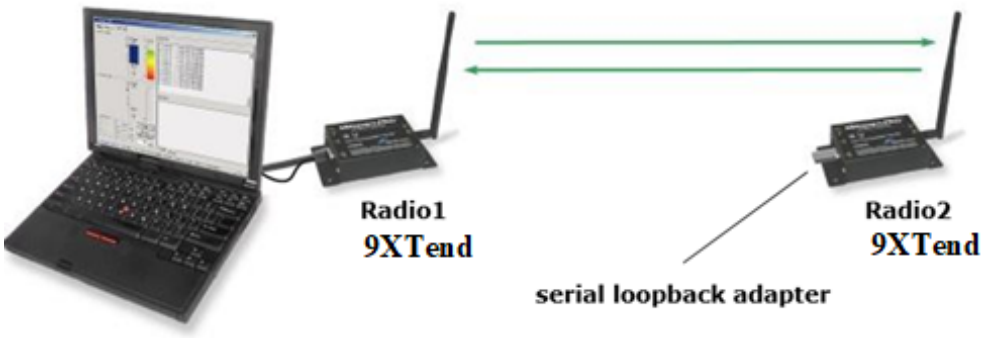

Figure 3.4 Radio communication using XTend RF Modem

For our thesis implementation, XTend RF Modem [49] has been used; one used in the IoT subsystem and other one at the base station. The AT commands have been used to configure

the Modems to support additional functionalities. The serial communications between a host (in our case Raspberry Pi 3) and an XTend RF Modem are dependent upon having matching baud rate, parity, stop bit, and the number of data bits settings. The Modem has Config (Configuration) Switch, I/O and Power LEDs, DB-9 Serial Port, RSS LEDs, Power Connector, DIP Switch and Antenna Port as external interfaces.

Figure 3.5 USB to RS232 cable used to connect Radio Modem with Raspberry Pi 3 [50]

The XTend RF Modem can be connected to host device with the help of standard DB-9 RS232 connector cable. Devices can also be connected directly through pin of XTend RF Modem, if the devices have support of RS 232 serial port, as shown in Figure 3.6.

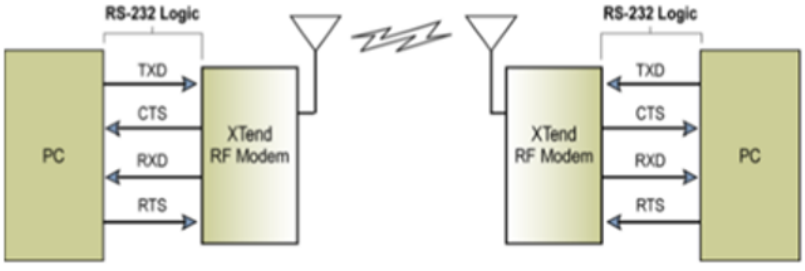

Figure 3.6 System Data Flow in RS-232 environment [49]

C. Satellite Modem for Data Communication

The RockBLOCK Mk2 [51] is a Satellite Modem for the Iridium Satellite network. RockBLOCK Mk2 has a built-in patch antenna and has support for external antenna connection. RockBLOCK Mk2 Modem is compatible with FTDI serial interface cable that is used to connect modem with other devices such as PC, Raspberry Pi etc. to establish serial communication between devices and modem. RockBLOCK Mk2 Modem needs 470 mA with 5V power supply when it startup and around 100 mA continuous current requirement for its normal operation Modem can also be set in sleep mode to save power consumption but, it still consumes 20 μA(very less current requirement) during sleep mode. RockBLOCK Mk2 Modem can lao be powered by external power source such as 3.7 Volt lipo battery that can supply up to 1500 mA. Although, Satellite Modem has a global coverage but, compared to the XTend Radio modem, data cost is very high as this device depends on the infrastructure, data plan and subscription services for message transmission. There are other downsides such as slower transmission time (up to 20 seconds to 1 minutes for a single message), high data cost for sending or receiving message, latency (up to 1 minutes) etc. Messages sent or received from RockBLOCK Mk2 Satellite Modem are transferred through the Internet, via an email-address or with HTTP (Hyper Text Transfer Protocol). In our design, the RockBLOCK Mk2 Satellite Modem is connected to Raspberry Pi 3 through USB to RS232 cable.

3.1.4 Power Subsystem

The role of the power subsystem is to provide and distribute power to each component of system, and manage the energy in the attached Radio, Satellite Modem and Gateway node

(Raspberry Pi 3). An external portable Lithium ion battery has been used to support power requirements

3.2 Data Processing

The proposed IoT system is comprised of sensor nodes, energy-efficient communication networks, and the IoT Gateway that transmits data to the base station. XTend RF Modem, RockBolock Iridium SBD Satellite Modem, Wi-Fi, Raspberry Pi 3 and cloud services are used to implement the proposed IoT system. Dweet.io [52], an open source cloud broker for prototyping an end-to-end solution is used and developed dashboard for visualizing IoT data on a cloud application.

Hardware components as described so far are configured to form an IoT system. IoT system along with software components is the base for the proposed IoT telemetry solution. The Software implementation and the data processing is explained in the following sections.

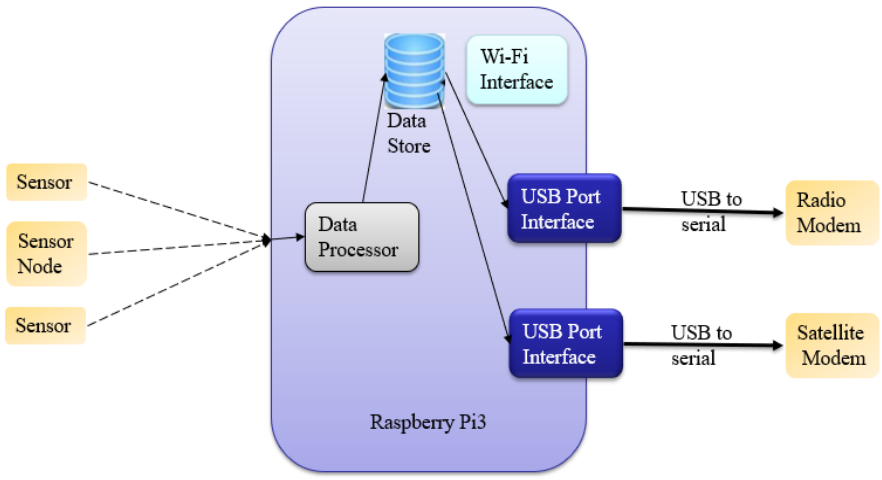

Figure 3.7 Data Flow Process in the Proposed IoT system

3.2.1 IoT Gateway Data Processing

Raspberry Pi 3 model B is used as a IoT Gateway for data processing, aggregation, storage and further visualization of sensor data on dashboard with the help of developed software programs and algorithms using python.

Figure 3.8 Raspberry Pi 3 model B used as an IoT Gateway [46]

Raspberry Pi 3 is powered by Cortex-A7 processor which is very high-performance processor. It is capable of running all kind of GNU/Linux distribution compatible software, packages and applications. Raspberry Pi 3 can also run Windows 10 application. Due to its reliability and large support community, OS Raspbian Jessie was used via a 16GB flash drive. The data aggregate node/IoT Gateway must receive large amounts of sensor data for processing and transmit them via Wi-Fi, Radio or Satellite link. Both tasks are relatively energy intensive tasks. The Raspberry Pi 3 requires a strict 5V regulated power supply with current draw up to 2A depending on application hardware.

An IoT Gateway has many functionalities. First and most important task of the Gateway is to transformation and normalization of the heterogeneous data coming from different sensors (data sources). The generated data sets can vary in structures and formats

such as JSON, CSV, XML etc. Some of the legacy nodes use proprietary protocols, while the contemporary ones may rely on JSON or CSV. Hence, the Gateway accumulates heterogeneous nature of data coming from multiple sensor nodes. Then, accumulated data is converted to a proper format that is required for next step of data processing.

Protocol transformation is the second important task of an IoT Gateway. Since sensor nodes are low powered and constraint devices that cannot use the Wi-Fi or Ethernet which require lots of power for data transmission. They use low-powered communication protocol for data transmission. IoT Gateway support multi-protocols communication for accepting inbound communication ie. sensor data sent by sensor nodes and outbound communication i.e. Gateway connection to cloud through remote API and using protocols such as MQTT, CoAP, REST, etc. [65] Gateway can also work for processing data and real-time monitoring.

Figure 3.9 shows the north bound and southbound protocol support architecture of an IoT Gateway.

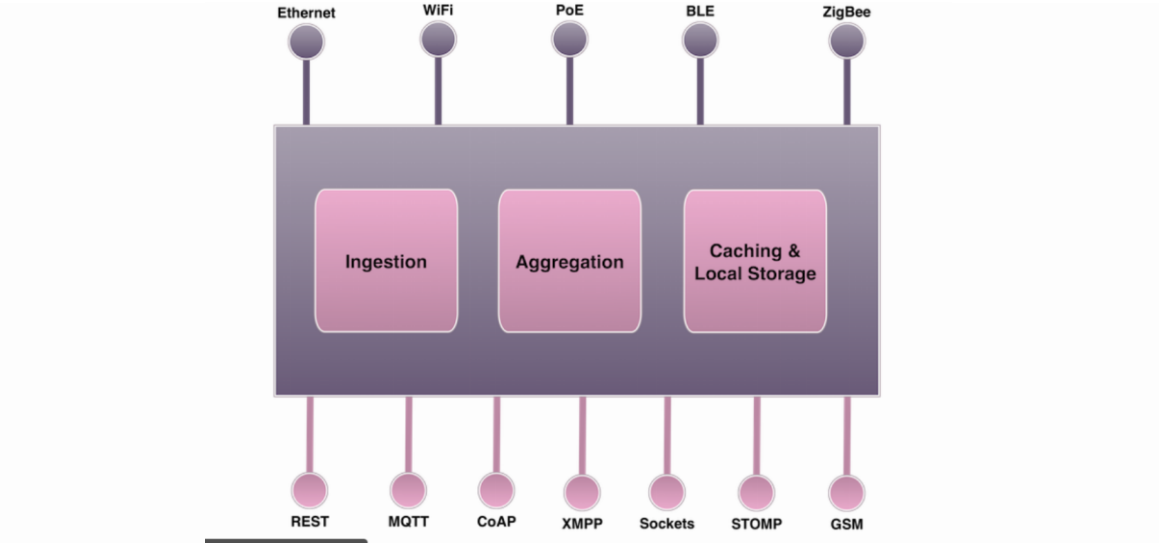

Figure 3.9 Block diagram of IoT Gateway with northbound and southbound protocol support [65]

Gateways behaves and act as an edge device which obscure the sensor nodes direct access from public internet and cloud [65]. Sensor nodes can make connection to the cloud and public internet through outbound protocols running at Gateway but, cannot be directly access by public internet. Hence, Gateways provides security and restricted access to internal sensor nodes. In this way Gateway also acts like firewall and router for internal sensor nodes network

Gateway helps in data transformation, aggregation and processing. Sensors can make a remote connection with an application running on cloud side referred as cloud Gateway. The local edge device running closely to sensor nodes is also known as a Field Gateway, as shown in Figure 3.10.

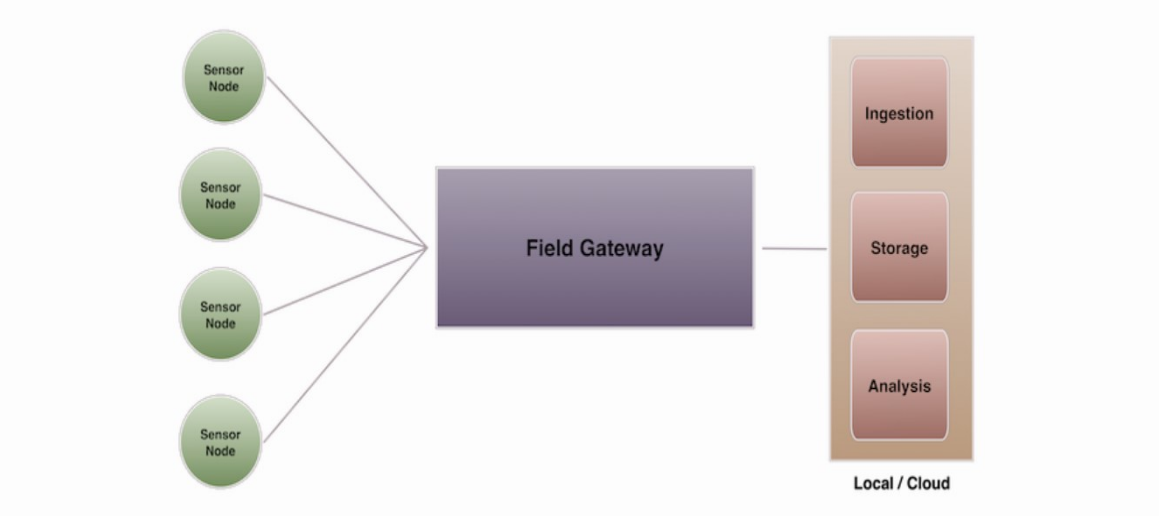

Figure 3.10 IoT Gateway as edge device between sensor nodes and cloud applications [65]

3.2.2 Data Transmission through Wi-Fi

Sensor data is stored locally into SQLite database, which is a light weight database installed onto the IoT Gateway. The IoT Gateway receives high volume of data from all sensors

and performs filtering and aggregation as per set rules. As Wi-Fi provides highest data rate with low cost so the data rate limiter algorithm selects high volume of data to be transmitted through Wi-Fi. A python based socket client program is developed to connect IoT system with remote Base station server. IoT Gateway performs SQL queries to select data from the SQlite DB installed onto Raspberry Pi 3 (Gateway node) and transmits it to the nearest base station server through Wi-Fi. The base station is further connected to Cloud network via internet backbone. AMQP is a publish/subscribe message mechanism, which is used to transmit the data to cloud application . IoT Gateway publishes topic to dweet.io which is a cloud broker. A freeboard dashboard application has been designed which subscribes to the topic published by IoT Gateway.

3.2.3 Data Transmission through XTend Radio

The selected Radio Modem has communication range up to 40 miles in outdoors line of sight and 3000ft in indoor. This Modem operates within the ISM 900MHz frequency band and with data throughput up to 115.2kbps. The transmit power can be adjusted from 1mW to 1W as per the application requirement. To avoid interference, it uses Frequency Hopping Spread Spectrum (FHSS) to hop to a new frequency on every data packet transmission.

The XTend Radio Modem's default configuration can be extended by using simple AT or binary commands. The 900-MHz Radio link as another mode of communication provide cost free solution for sending data to base station. It is cheapest mode of communication compared to Satellite, Cellular or Wi-Fi.

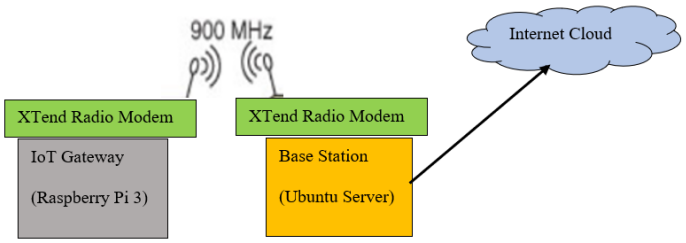

Figure 3.11 Data Transmission between XTend Radio Modems at IoT Gateway on Mobile Terminal and Base Station

Figure 3.11 shows RF communication between Gateway on Mobile Terminal and the Base Station. The Radio modem is serially connected at IoT Gateway and at Base Station. It is configured with source and destination address using XCTU software to transmit data. The python program running at IoT Gateway node transmits sensor data over the serial port through transmitter Radio Modem and same sensor data appears on serial port of the destination Radio Modem's RX line connected at base Station Server. Ubuntu Server at the Base station hosts Apache Webserver and make requests to the IoT Gateway to send sensor data periodically. When request is received by XTend RF Radio modem at IoT Gateway, it transmits the sensor data to the Base Station.

3.2.4 Data Transmission through Satellite

The RockBLOCK Mk2 is an Iridium 9602 Short Burst Data (SBD) Satellite Modem which can send and receive short messages from anywhere. Iridium has 66 Satellites in orbit around earth and provides a global coverage. Figure 3.12 shows the working of RockBlOCK Mk2 Iridium SBD Satellite Modem. The Modem transmits data to the Iridium constellation which transmits it to Iridium Gateway Ground Station. RockBLOCK server

receives data from the Iridium Servers and send it to the user's application i.e. web application, email etc.

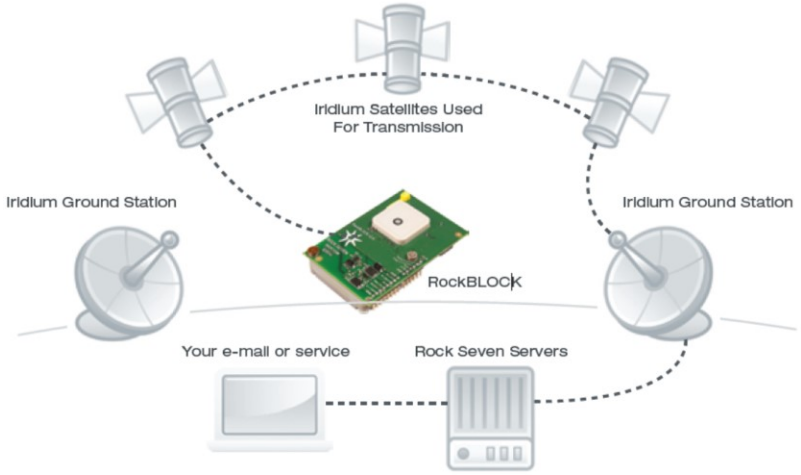

Figure 3.12 Working of RockBLOCK Mk2 Iridium SBD Satellite Modem [51]

Figure 3.13 RockBLOCK Mk2 Iridium 9602 Satellite Modem [51]

a) **MO-SBD Message Transmissions Overview**: MO-SBD is mobile unit originated Short bust data message. These short-burst messages originate from the Raspberry Pi 3 with Iridium SBD Modem and are then sent to the Iridium Gateway and finally to the destination e-mail address and web services as shown in Figure 3.14. The Raspberry Pi 3 acting as Gateway node receives a status signal for success or failure of message

transmission and accordingly controls the transmission process. The transmission has been executed with the help of standard Modem “AT” commands [3].

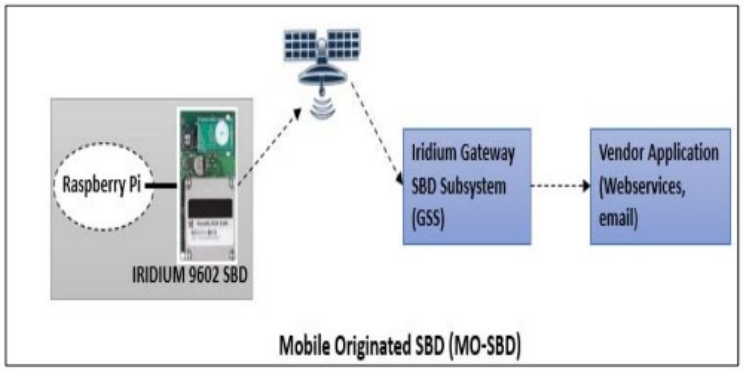

Figure 3.14 Mobile originated SBD [3]

b) **MT -SBD Message Transmissions Overview:** MT-SBD is mobile terminated short bust data message. These short-burst messages transmit from Iridium Gateway to Iridium SBD Modem attached with Raspberry Pi 3, which is identified by IMEI number of the SAT Modem as shown in Figure 3.15 After successful transmission of the SBD message an acknowledgement is sent back. The Iridium SBD Modem retrieves MT messages. These MT messages are decoded by connected Raspberry Pi 3 from the Iridium SBD Modem through a set of “AT” commands [3].

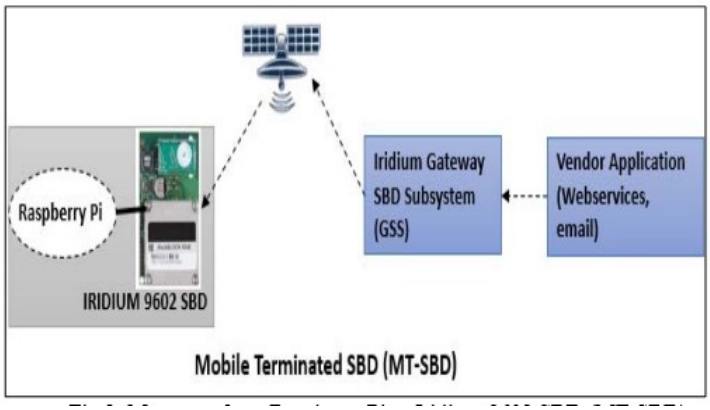

Figure 3.15 Mobile terminated SBD [3]

3.3 Overview and Architecture of VHD Mechanism

The VHD algorithm continuously monitors the wireless interface of Raspberry Pi 3 and serial interface of the RockBLOCK Mk2 Satellite (SAT) Modem. In the handover decision process, various criteria are defined for selecting available interface (Wi-Fi, Radio or SAT) based on Network availability, Signal strength, cost of service, Network condition and system performance. The algorithm is designed to calculate the best network by evaluating multiple criteria and assigning the priority. If Wi-Fi mode of communication is selected then it streams all collected sensor data by the IoT system to the cloud application. If Wi-Fi is not available or the system senses poor Wi-Fi signal strength, then the system does a vertical handover to the Radio or Satellite connection as per calculated priority ranking based on Multi Criteria. When the system again detects strong Wi-Fi secure connection for a stable period of time, then it again calculates priority ranking and handovers from Satellite to Wi-Fi or Radio interface as per priority. In undesirable conditions like when Wi-Fi interface is down, Radio interface is also down, and Satellite Modem signal strength is also poor or Modem waiting for coming in activation state, the handover algorithm is designed to store the data packets into buffer to avoid packet loss which are supposed to be transmitted by using any medium and starts transmitting when any of the available interface is successfully connected. Short messages are then sent to the web servicer running at base station . The handover process occurs in the following steps:

a) **Network discovery** – responsible for identifying networks that are in range of the device. It is usually done by scanning the different available interfaces of heterogeneous networks

b) **Network selection** – a choice is made by the terminal or the network to select best available network to which the terminal can be connected.

c) **Handover** – the actual process of switching the data session from one network to another, which is highly dependent on optimal network selection among available heterogeneous networks based on multiple criteria (data rate, quality of RSS, delay, cost, etc).

3.4 Interface Scanning Module

This section describes the interface scanning module running on IoT Gateway (Raspberry Pi 3), which scan all the three attached interfaces and select those interfaces which are active and can connect to the network. In the experiment testbed, there are three heterogenous networks Wi-Fi, Radio and Satellite. Radio Modem and Satellite are connected through serial interfaces while, Wi-Fi is wireless interface. This module performs continuous network scanning, i.e. scanning of various interfaces and connection to heterogeneous networks available in the coverage area and then updates the list of all available networks. If a network is not reachable then it is removed from the list of active network interface. After getting the list of available networks then calculation of quality of received signal and other parameters is done by other modules which will be discussed later in this chapter.

3.4.1 Interface Scanning Algorithm

This section describes the algorithm for network interface scanning for Wi-Fi, Radio link and Satellite link. The algorithm is implemented using python 2.7 language, as shown in in Algorithm 3.1, is periodically invoked to check the status of all the interfaces. Algorithm

3.1 keeps on updating only the active interfaces and pruning inactive interfaces. Hence, the calculation for quality of RSS and other parameters for handover decision are done only for active interfaces, which is described later in chapter.

Algorithm 3.1: Scanning Interfaces and returning the active interfaces

---

Input: I: Interface set, i: selected Interface
Output: I_{Active}: set of active interfaces

```
1    I : set of all interfaces
2    i : selected interface, where i ∈ I
3    I_Active: set of Active interfaces
4    foreach i ∈ I do
5        Check the status of interface i
6        if interface i is Active then
7            Add the interface to list I_Active
8        end if
9    end for
```

Algorithm 3.1 removes the inactive network interfaces from the list of all the heterogeneous interfaces, i.e., Wi-Fi, Radio and Satellite. After completing the scan of network interfaces, selecting active interfaces, other parameters such as RSS, Data Rate, etc. are calculated for those active interfaces to decide whether a handover should be performed or not. The handover decision is made by handover decision module.

3.5 Handover Decision Module

This module consists of the main handover decision algorithm. This section contains two methods for deciding the vertical handover. Firstly, conventional RSS Quality based VHD method is described which only takes RSS Quality criterion into consideration for VHD.

Secondly, the proposed MCVHD method is described which is based on multi criteria for VHD.

3.5.1 Conventional RSS Quality based VHD Method

The conventional handover decision method is based on the received signal strength (RSS) Quality. The procedure to calculate the RSS values for network like the Radio link and the Satellite link are different from that of Wi-Fi link, therefore, separate methods are used for each of the three networks to measure the RSS Quality. For Wi-Fi and Radio network RSS value in dBm is measured and then converted the RSS value in terms of RSS Quality percentage for making all the RSS Quality value at common scale of 0% to 100%. The Satellite network gives RSS Quality value in scale of 0 to 5, hence it is also converted to RSS Quality percentage from (scale 0 value) 0% to (scale 5 value) 100%. This Quality conversion makes Wi-Fi, Radio and Satellite Network to be evaluated on the basis of RSS Quality percentage (0%-100%), where 50% of RSS Quality is taken as threshold limit for handover decision. The RSS values for all the active network interfaces are calculated based on algorithms presented in Algorithm 3.2, Algorithm 3.3 and Algorithm 3.4.

Algorithm 3.2 presents an algorithm to calculate RSS Quality of Wi-Fi network. The algorithm measures $RSS_{Wi\text{-}Fi}$ value by sending Linux command *iwconfig wlan0* periodically and coverts the received $RSS_{Wi\text{-}Fi}$ (dBm) into $RSS_{Wi\text{-}Fi}$ Quality (percentage). This algorithm calculates the $RSS_{Wi\text{-}Fi}$ Quality by using equation presented in eq (1).The $RSS_{Wi\text{-}Fi}$ value is measured in units of dBm. Therefore, the conversion between $RSS_{Wi\text{-}Fi}$ Quality (percentage) and dBm [63] is as follows:

$$RSS_{Wi\text{-}Fi}\,\text{Quality} = 2\times(RSS_{Wi\text{-}Fi} + 100) \quad \text{...eq (1)}$$

where $RSS_{Wi\text{-}Fi}$:[-100 dBm to -50 dBm]

If the $RSS_{Wi\text{-}Fi}$ Quality is "0%" that indicates Wi-Fi has very poor RSS. Similarly, if the $RSS_{Wi\text{-}Fi}$ Quality is "100%" that indicates Wi-Fi has very strong received RSS and best network for efficient network selection. The value between 0% and 100% are decided and calculated based on eq (1) mentioned in procedure. The algorithm for calculating $RSS_{Wi\text{-}Fi}$ Quality of Wi-Fi network is presented in Algorithm 3.2

Algorithm 3.2: Calculate $RSS_{Wi\text{-}Fi}$ Quality of Wi-Fi network

---

Input: Linux command (*iwconfig wlan0*)
Output: $RSS_{Wi\text{-}Fi}$ Quality

```
1   Send Linux command to receive RSS_Wi-Fi        //command–> iwconfig wlan0
2   if (Received RSS_Wi-Fi <= -100) then
3       RSS_Wi-Fi Quality = 0;
4   else if (Received RSS_Wi-Fi >= -50) then
5       RSS_Wi-Fi Quality = 100;
6   else
7       RSS_Wi-Fi Quality = 2 * (Received RSS_Wi-Fi + 100);
8   end if
9   return RSS_Wi-Fi Quality
```

Algorithm 3.3 is used to calculate the RSS_{Sat} Quality of the Satellite link. At first, the AT command (CSQ) is send periodically through the serial port of RockBLOCK Mk2 Iridium Satellite Modem. In response of the AT command, the Quality value in the range of 0 to 5 is received. After receiving Quality, it is then scaled in terms of percentage (0%-100%) and stored in variable RSS_{Sat} Quality. Algorithm 3.3 is performed when the Satellite interface is selected as an active interface.

Algorithm 3.3: Calculate RSS_{Sat} Quality of Satellite network

Input: AT command (AT+CSQ) through the serial port using RS-232 Protocol
Output: RSSSat Quality

```
1   Send AT command to receive RSSSat Quality          //command–> AT+CSQ
2   if (Received RSSSat Quality == 0) then
3       RSSSat Quality = Received RSSSat Quality * 20
4       Display Signal Status: No Signal
5   else if (1<= Received RSSSat Quality <= 2) then
6       RSSSat Quality = Received RSSSat Quality * 20
7       Display Signal Status: Weak Signal Strength
8   else if (2 <= Received RSSSat Quality <= 4) then
9       RSSSat Quality = Received RSSSat Quality * 20
10      Display Signal Status: Moderate Signal Strength
11  else if (Received RSSSat Quality ==5)
12      RSSSat Quality = Received RSSSat Quality * 20
13      Display Signal Status: Strong Signal Strength
14  end if
15  return RSSSat Quality
```

RSS_{Radio} Quality of the 900 MHz Radio link is calculated using Algorithm 3.4. At first, the AT command (ATRS) is send periodically through the serial port of XTend RF Modem (Radio Modem) to monitor the signal strength and connectivity of 900 MHz Radio link between the mobile IoT Gateway (Transmitter Radio Modem) and the base station (Receiver Radio Modem). In response of the AT command, signal strength of Radio link ($RSS_{Radio)}$ is received. The received RSS_{Radio} (dBm) is then compared with fade margin to convert in terms of percentage and stored in variable RSS_{Radio} Quality.

Algorithm 3.4: Calculate RSS Quality of Radio network

Input: AT command (AT+ATRS) through serial port using RS-232 Protocol
Output: RSS_{Radio} Quality

```
1   Send the AT command to receive RSS_Radio      //AT command –> AT + ATRS
2   if (Received RSS_Radio > 30 fade margin) then
3       RSS_Radio Quality = 100
4       Display Signal Status: Very Strong Signal
5   else if (20 fade margin < Received RSS_Radio ≤ 30 fade margin) then
6       70 < RSS_Radio Quality < 100
7       Display Signal Status: Strong Signal
8   else if (10 fade margin < Received RSS_Radio ≤ 20 fade margin) then
9       40 < RSS_Radio Quality ≤ 70
10      Display Signal Status: Moderate Signal
11  else if (0 fade margin <Received RSS_Radio ≤ 10 fade margin) then
12      0 < RSS_Radio Quality ≤ 40
13      Display Signal Status: Weak Signal
14  else
15      RSS_Radio Quality = 0
16      Display Signal Status: No Signal
17  end if
10  return RSS_Radio Quality
```

After measuring RSS Quality of the network, Algorithm 3.5 presented in flow chart shown in Figure 3.16, is used for vertical handover decision making. Algorithm 3.5 selects best network among Wi-Fi, Radio and Satellite network, based on RSS Quality. The threshold of each network is considered to be 50% of RSS Quality and the vertical handover takes place if network reaches below the threshold.

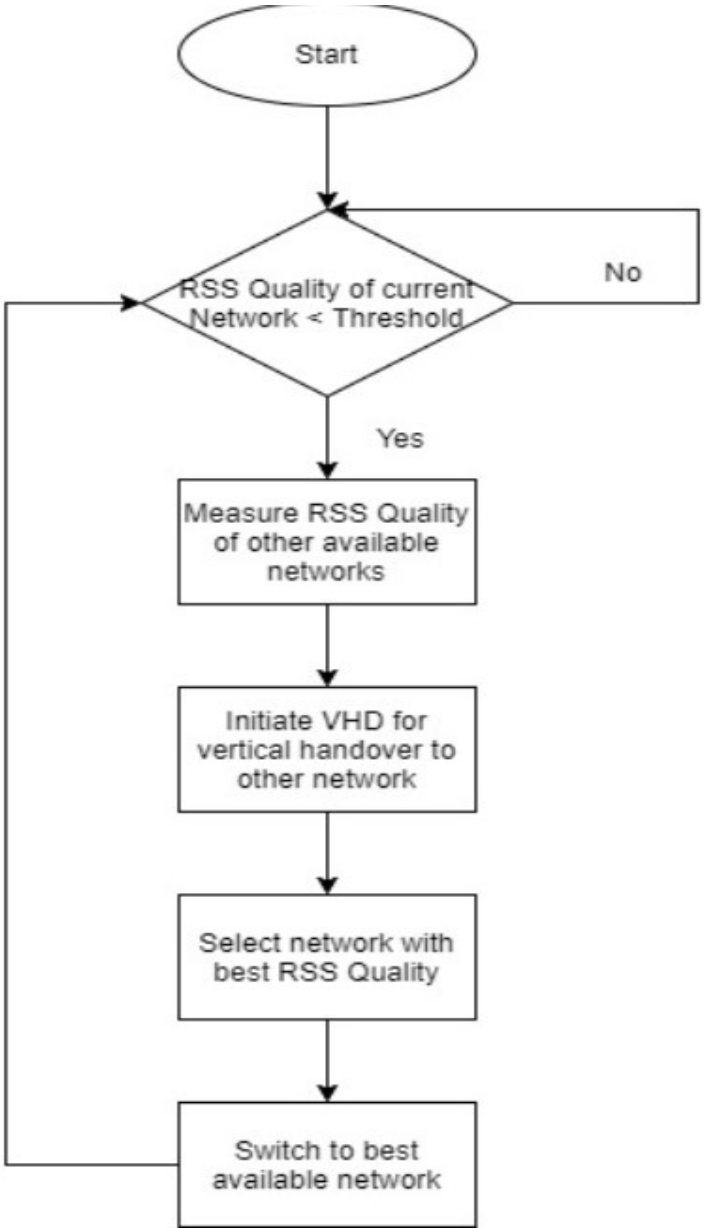

Figure 3.16 Conventional RSS Quality based VHD algorithm (Algorithm 3.5).

A conventional RSS Quality based VHD method for vertical handover among heterogeneous networks such as Wi-Fi, Radio, Satellite, is explained with the help of flow chart in Figure 3.16. In RSS Quality based VHD method, MT periodically monitors for the RSS Quality of the current network to detect the condition for vertical handover. When the

RSS Quality of current network is going below the threshold value of RSS Quality, the other available heterogeneous networks are scanned by the MT. Any other network having best RSS Quality is selected by VHD and finally the network connection is handed to the chosen best available network, thus vertical handover completes.

3.5.2 Multi Criteria based VHD Method

The proposed handover decision strategy referred as MCVHD, is used to calculate the priority ranking of the available networks based on multi-criteria such as link quality, price, bandwidth, battery consumption, etc. During the handover decision, the available networks are ranked based on multi-criteria and user preference weights. The network ranked with highest priority is selected as a target network to handover.

3.5.2.1 Vertical Handover Decision (VHD) Method based on Multi Criteria

VHD makes a selection decision among the available networks to handover to the best network based on analyzing multi-criteria. This type of method based on multi-criteria falls under the domain of Multi-Attributed Decision Making (MADM) method [39]. This thesis only considers three heterogenous networks for vertical handover i.e. Wi-Fi, Radio and Satellite network, denoted by N1, N2 and N3 respectively. The multi-criteria considered for vertical handover are RSS Quality, data service cost, data rate, latency, reliability, network coverage range, power consumption, mobile terminal velocity which are denoted by $P1, P2, P3, P4, P5, P6, P7$ and $P8$, respectively as shown in Table 3.1.

Table 3.1 Notation for multi-criteria parameters

Parameters	Notation
RSS Quality (0%-100%)	P1
Data Service Cost (per 100Kb) in CAD	P2
Data Rate (Mbps)	P3
Network Latency (Linguistic term)	P4
Reliability (Linguistic term)	P5
Power Requirement (mW)	P7
MT Velocity (km/h)	P8

The multi-criteria can be concisely expressed in the form of decision matrix D, where the capabilities of each network, N1 (Wi-Fi), N2 (Radio) and N3 (Satellite) are presented with criteria (P1, P2,…..P8). The criterion like latency (P4) depends on the routing to the destination terminal and cannot be determined prior to evaluate the network, hence a linguistic term could be used for such criterion. An average value of latency can also be used if algorithm has prior knowledge of the received latency of all the networks. Latency (P4) and reliability (P5) , whose value cannot be precisely determined are taken as linguistic terms as Low, Medium and High.

$$D = \begin{array}{c} \\ N1 \\ N2 \\ N3 \end{array} \begin{array}{c} \begin{array}{cccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 & P8 \end{array} \\ \begin{bmatrix} 75 & 0.005 & 50 & Low & Low & .03 & 500 & 20 \\ 80 & 0 & 0.0091 & Medium & Medium & 5 & 2000 & 20 \\ 70 & 215 & 0.0183 & High & High & 1000 & 500 & 20 \end{bmatrix} \end{array}$$

If any network is not reachable then it is excluded from the decision matrix D.

The preference on handover criteria is modelled as weights assigned by the user on the criteria as shown below by the weight vector W.

$$W = [High \quad Low \quad Medium \quad Medium \quad High \quad Medium \quad Low \quad Medium]$$

This type of imprecise data cannot be efficiently handled by any of the MADM methods, hence the proposed MCVHD method enhances MADM approach by using fuzzy logic to convert each linguistic term into crisp numbers as per fuzzy number conversion scale [41].

Chen and Hwang [41] have proposed different conversion scales with different number of linguistic terms. The same linguistic term in different conversion scales can have different crisp values. E.g., when six linguistic terms, very low, low, fairly low, fairly high, high and very high are used, the term high will be converted to the crisp number 0.75. This thesis uses one of the fuzzy number conversion scale in our MCVHD algorithm to calculate priority ranking of Wi-Fi, Radio and Satellite network as shown in Figure 3.17 where μ(x) is degree of membership and x axis represents Fuzzy number range of very low, low, medium, high and very high.

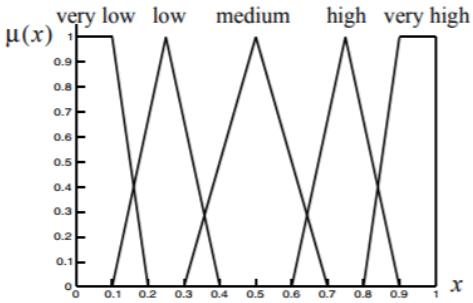

Figure 3.17 Linguistic term to fuzzy number conversion scale [41]

The proposed MCVHD algorithm uses TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) [64] technique as one of the MADM methods to develop seamless vertical handover as IoT solution. TOPSIS technique is enhanced by using Fuzzy logic for converting linguistic terms to numerical values. TOPSIS technique is based on

the principle that the chosen alternative should have the shortest distance from the ideal solution and the farthest distance from the negative-ideal solution.

3.5.2.2 Converting linguistic terms using Fuzzy Logic

The multi-criteria for Wi-Fi, Radio and Satellite network are converted into decision matrix D, as presented in M (1). Decision matrix D includes linguistic terms for criteria latency (P4) and reliability (P5)

$$D = \begin{array}{c} \\ N1 \\ N2 \\ N3 \end{array} \begin{array}{c} \begin{array}{cccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 & P8 \end{array} \\ \begin{bmatrix} 75 & 0.005 & 50 & Low & Low & .03 & 500 & 20 \\ 80 & 0 & 0.0091 & Medium & Medium & 5 & 2000 & 20 \\ 70 & 215 & 0.0183 & High & High & 1000 & 500 & 20 \end{bmatrix} \end{array} \quad \ldots M\,(1)$$

Decision matrix D as presented in M (1) is converted using fuzzy number conversion scale, see Figure 3.17. The resultant decision matrix D after converting linguistic terms is presented in M (2) as shown below.

$$D = \begin{array}{c} \\ N1 \\ N2 \\ N3 \end{array} \begin{array}{c} \begin{array}{cccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 & P8 \end{array} \\ \begin{bmatrix} 75 & 0.005 & 50 & 0.090 & 0.090 & .03 & 500 & 20 \\ 80 & 0 & 0.0091 & 0.250 & 0.250 & 5 & 2000 & 20 \\ 70 & 215 & 0.0183 & 0.950 & 0.950 & 1000 & 500 & 20 \end{bmatrix} \end{array} \quad \ldots M\,(2)$$

User defined weight vector W, as presented in W (1), for multi-criteria are also converted to numbers and normalized so that the sum is equal to 1. After conversion the normalized weight vector W′ for multi-criteria parameters is presented in W (2).

$$W = [\text{High} \quad \text{Low} \quad \text{Medium} \quad \text{Medium} \quad \text{High} \quad \text{Medium} \quad \text{Low} \quad \text{Medium}] \quad \ldots W\,(1)$$

$$W' = [0.222 \quad 0.055 \quad 0.111 \quad 0.111 \quad 0.222 \quad 0.111 \quad 0.055 \quad 0.111] \quad \ldots W\,(2)$$

After the linguistic terms are converted into numbers, the TOPSIS technique is used to calculate the ranking of the available networks.

3.5.2.3 Proposed MCVHD method based on enhancing TOPSIS Technique by using Fuzzy Logic

Below is the procedure for the proposed MCVHD algorithm based on enhancing TOPSIS technique using fuzzy logic:

a) Step 1: Construct decision matrix using fuzzy logic approach mentioned in section 3.5.2.2. The decision matrix D as presented in M (2) is composed of different available heterogeneous networks N1, N2 and N3 by using the different priority based criteria (P1, P2, P3,….P8) .

$$D = \begin{array}{c} \\ N1 \\ N2 \\ N3 \end{array} \begin{array}{c} \begin{array}{cccccccc} P1 & P2 & P3 & P4 & P5 & P6 & P7 & P8 \end{array} \\ \begin{bmatrix} 75 & 0.005 & 50 & 0.090 & 0.090 & .030 & 500 & 20 \\ 80 & 0 & 0.009155 & 0.250 & 0.250 & 5 & 2000 & 20 \\ 70 & 215 & 0.01831 & 0.950 & 0.950 & 1000 & 500 & 20 \end{bmatrix} \end{array} \quad \ldots \text{M (2)}$$

Where p_{ij} is the value of the i^{th} network (N) and j^{th} criteria (P) in the decision matrix D.

b) Step 2: A normalized decision matrix D' is constructed to transform dimensional attributes into normalized way by Euclidean normalization. Euclidean normalization converts all attributes into dimensionless units which makes easy comparison across multi-criteria, presented by decision matrix M (3). Other normalization technique like Linear Scale Transformation is inefficient as calculated scale transformation is not proportional to outcome [66].

Each element of the normalized decision matrix D' is calculated as r_{ij} . The calculation of r_{ij} is given below by eq (2):

$$r_{ij} = {p_{ij}}\Big/{\sqrt{\sum_i^n {p_{ij}}^2}}, i = 1,\dots n, j = 1,\dots m. \quad \text{...eq (2)}$$

Where, r_{ij} represents each element of the normalized decision matrix.

p_{ij} is the value of the i[th] network and j[th] criteria in the decision matrix.

n represents the available network

m represents the multi-criteria

As there are 3 heterogenous networks and 8 network criteria in the proposed MCVHD so each element of the normalized decision matrix D' is calculated as below presented in eq (3)

$$r_{ij} = {p_{ij}}\Big/{\sqrt{\sum_i^3 {p_{ij}}^2}}, i = 1,..3, j = 1,\dots 8. \quad \dots\text{eq (3)}$$

n = 3, (Wi-Fi, Radio and Satellite).

m = 8, (RSS Quality, price, data rate, latency, reliability, network coverage range, power consumption, mobile terminal velocity).

Calculated normalized decision matrix D' is given below

$$D' = \begin{matrix} N1 \\ N2 \\ N3 \end{matrix} \begin{bmatrix} 0.565 & 0.00002 & 0.999 & 0.091 & 0.0912 & 0.0001 & 0.235 & 0.57 \\ 0.615 & 0 & 0.0001 & 0.253 & 0.2534 & 0.0049 & 0.942 & 0.57 \\ 0.538 & 0.999 & 0.0003 & 0.963 & 0.963 & 0.9999 & 0.235 & 0.57 \end{bmatrix} \quad \dots\text{M(3)}$$

c) Step 3: Construct the weighted normalized decision matrix U using cross product of normalized decision matrix D' as presented in M (3) and weight vector W' as presented in W (2). Each element u_{ij} of the weighted normalized decision matrix U can be calculated by multiplying the normalized decision matrix r_{ij} with its associated weight in normalized weight vector W' as presented below in eq (4):

$$U = W' \times D' \quad \text{... eq (4)}$$

The calculated weighted normalized decision matrix U is given below as presented in M(4)

$$U = \begin{matrix} N1 \\ N2 \\ N3 \end{matrix}\begin{bmatrix} 0.125 & 0.000 & 0.110 & 0.010 & 0.020 & 0.000 & 0.012 & 0.064 \\ 0.027 & 0 & 0.0001 & 0.028 & 0.056 & 0.001 & 0.051 & 0.064 \\ 0.119 & 0.054 & 0.0001 & 0.106 & 0.213 & 0.111 & 0.013 & 0.064 \end{bmatrix} \quad \text{…M(4)}$$

d) Step 4: Determine the positive ideal solution $S^+$and the negative ideal solution S^- by using eq (4) and (5), where J is associated with the benefit criteria and is J^*associated with the cost criteria.

$$S^+ = [u_1^+ \quad u_2^+ \quad u_3^+ \quad u_4^+ \quad u_5^+ \quad u_6^+ \quad u_7^+ \quad u_8^+]$$

$$= \left\{\left({}^{max}_{i}u_{ij} | j \in J\right), \left({}^{min}_{i}u_{ij} | j \in J^*\right) \middle| \begin{matrix} i = 1,..3 \\ j = 1,2,..8 \end{matrix}\right\} \quad \text{…eq (5)}$$

$S^+ = [0.1254 \quad 0 \quad 0.1108 \quad 0.0101 \quad 0.2137 \quad 0.1109 \quad 0.0129 \quad 0.0640]$ …M (5)

$$S^- = [u_1^- \quad u_2^- \quad u_3^- \quad u_4^- \quad u_5^- \quad u_6^- \quad u_7^- \quad u_8^-]$$

$$= \left\{\left({}^{min}_{i}u_{ij} | j \in J\right), \left({}^{max}_{i}u_{ij} | j \in J^*\right) \middle| \begin{matrix} i = 1,..3 \\ j = 1,2,..8 \end{matrix}\right\} \quad \text{…eq (6)}$$

$S^- = [0.0278 \quad 0.0549 \quad 0.00001 \quad 0.1068 \quad 0.0202 \quad 0.0 \quad 0.0518 \quad 0.0640]$ …M (6)

e) Step 5. Measure distance from the ideal solution. The Euclidean alternative distances, in relation to the positive and negative ideal solutions are determined by using eq (7) and eq (8) for each alternative.

$$D_i^+ = \sqrt{\sum_i^m (u_{ij} - u_j^+)^2} \quad , i = 1, \ldots, n \qquad \ldots \text{eq (7)}$$

$$D_i^- = \sqrt{\sum_i^m (u_{ij} - u_j^-)^2} \quad , i = 1, \ldots, n \qquad \ldots \text{eq (8)}$$

Calculate the separation of each alternative from the ideal solution, and the negative ideal solution, using the eq (7) and eq (8), and the result is presented as matrix in M (7) and M (8) respectively for Wi-Fi, Radio and Satellite.

$$D_i^+ = \sqrt{\sum_i^8 (u_{ij} - u_j^+)^2} \quad , i = 1, \ldots, 3$$

$$= [0.2230 \; 0.2462 \; 0.1571] \qquad \ldots \text{M (7)}$$

The separation from the negative ideal alternative is given by:

$$D_i^- = \sqrt{\sum_i^6 (u_{ij} - u_j^-)^2} \quad , i = 1, \ldots, 3$$

$$= [0.2830 \; 0.1462 \; 0.1976] \qquad \ldots \text{M (8)}$$

f) Step 6. The relative closeness R_i of each alternative, i={1,2,....,n} is the relative closeness from the ideal solution and is determined by below eq (9)

$$R_i = \frac{D_i^-}{D_i^+ + D_i^-} \quad , i = 1, \ldots . n \qquad \text{Where, } 0 < R_i < 1 \qquad \ldots \text{eq (9)}$$

Using the eq (9) to calculate relative closeness to the ideal solution for Wi-Fi, Radio and Satellite network

$$= [0.578\ 0.429\ 0.345] \quad \text{...M (9)}$$

g) Step 7. Networks ranking: The best network is according to the descending order of the relative closeness values Ri of the alternative. Hence, the network with the highest value of Ri is selected as an optimal network for handoff in available heterogeneous networks.

3.6 Rate of Data Transmission

This section consists the pseudo code for the algorithm for the rate of data transmission based on selected interface among the available active heterogeneous networks. The Algorithm 3.6, mainly responsible for limiting the data rate as per the selected network interface decided on above algorithms as discussed previously. So, the Algorithm 3.6, is highly dependent on previous algorithms 1, 2, 3 and 4 which scans the network interfaces. The Algorithm 3.6 contains a function UpdateDataRateLimit() which is periodically invoked when any new interface is selected. This function invoke the other functions such as SetWi-FiDataRate(), SetRadioDataRate() and SetSatDataRate() based on the selection of Wi-Fi, Radio or Satellite network respectively. After calculating the data rate based on selected interface then new data rate is updated by the UpdateDataRateLimit() for that selected new interface. Rate limiter function will minimize the data cost and save money by limiting data rate because the cost of sending the data through Satellite is very high compared to Wi-Fi and Radio link. Second advantage, this function is executed only when new interface is selected during handover. So, this will reduce memory consumption and

run time and its beneficial for small IoT devices which have less processing power as well as memory.

Algorithm 3.6: Calculation of Data Rate based on Network Interface Selection

Input: I: Interface Set, i: Selected Interface
Output: R: Data Rate, $R_{Wi\text{-}Fi}$, R_{Radio} R_{Sat}

```
1   I is set of all interfaces
2   i is selected active interface, where i ∈ I
3   R: Data Rate
4   begin
5       foreach i ∈ I do
6           if (i == Wi-Fi) then
7               // Update Data Transfer Rate for Wi-Fi interface
8               R_Wi-Fi = SetWi-FiDataRate()
9               R = R_Wi-Fi
10          else if (i == Radio) then
11              // Update Data Transfer Rate for Radio interface
12              R_Radio = SetRadioDataRate()
13              R = R_Radio
14          else if (i == Satellite) then
15              // Update Data Transfer Rate for Satellite interface
16              R_sat = SetSatDataRate()
17              R = R_Sat
18          end if
19      end for
20  return R
21  end
```

3.7 Chapter Summary

This thesis proposes efficient multi-criteria based VHD algorithm for Mobile IoT to handover between Wi-Fi, Radio and Satellite network. The proposed algorithm is based on TOPSIS technique which is a MADM based method. Since any of the MADM methods

cannot work with network criteria which have imprecise data value, hence the proposed thesis extends the TOPSIS technique by using Fuzzy logic to convert imprecise data into crisp numbers.

To evaluate the performance of the proposed MCVHD algorithm for IoT system, an experimental tested has been designed. The testbed is prepared using JY901 sensors which measure acceleration, magnetic field, orientation and angular velocity, a Raspberry Pi 3 which acts as an IoT gateway to collect sensor data, process and send it to the base station. The data is send to the base station using three heterogenous networks Wi-Fi, Radio and Satellite. Wi-Fi is built-in Raspberry Pi 3, while for Radio XTend Digikey Radio Modems are used as an transmitter at Mobile IoT and as receiver at base station. For satellite communication, RockBLOCK Iridium SBD Satellite modem is used.

Another algorithm is also designed for rate limiter network function. This algorithm works with MCVHD application and changes the rate of data transmission depending on the network selected for handover. The rate limiter network function reduces the service cost by reducing data rate whenever MCVHD selects satellite network due to high data service cost.

Chapter 4 Proposed Containerized IoT Solution using Docker Technology

Considering multiple criteria for designing a VHD algorithm is helpful for selecting the best available network. This chapter presents the benefits of developing components as microservices and explains the design of the proposed IoT solution as microservices architecture in Section 4.1. A container technology makes services to be independent and easy deployable. Using this advantage of container technology, Section 4.2 presents the microservices developed for proposed IoT solution as containerized solution. Containerization makes easier and more effective in software development, deployment of the MCVHD algorithm that are quick and effective in order to select best available network for power constraint IoT devices. Section 4.3 presents the Docker Compose Architecture for orchestration and management of microservices under a multi-container environment. Section 4.4 summarizes the design and implementation overview of the proposed containerized IoT solution.

4.1 Microservice Architecture

Microservices architecture is an approach of separating services into functional components that can interact via programming interfaces [53]. The proposed IoT solution is designed as microservices and reason behind using microservices architecture is due to the following benefits:

1) There is a negligible effect on the entire application when any microservice fails.

2) Managing independent functional components of the microservice architecture is relatively easy.

3) Since the functional components are independent, hence, adding, removing or changing any service is easier without affecting the entire application.

4) It is easy to deploy, replicate in all kinds of environments such as production, staging, and testing.

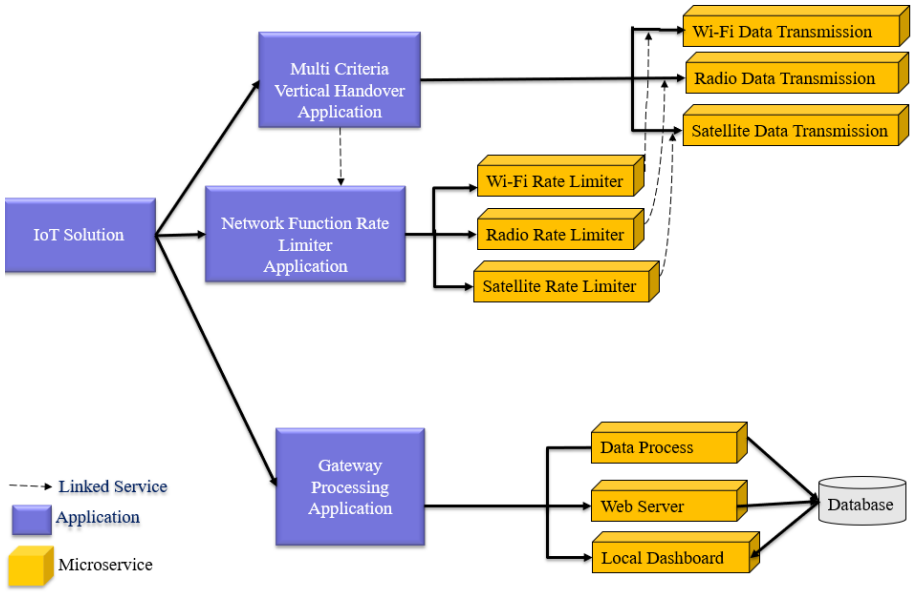

Figure 4.1 Microservice Architecture in the proposed IoT solution

Figure 4.1 presents the proposed IoT solution as microservices. The MCVHD Application is developed using microservices for Wi-Fi, Radio, and Satellite communications. Another application, Rate Limiter, is also developed as microservices where different data rate services are developed for data transmissions through Wi-Fi, Radio and Satellite interfaces. The Rate Limiter application and services are linked to the VHD application. For example, when the VHD application decides to handover to the Wi-Fi network, it uses the Wi-Fi microservice along with the rate limiter Wi-Fi microservice to provide data transmissions with the rate defined. There is another application, Gateway

Processing with separate microservices for Data Processing, Web Server and IoT Dashboard. Microservices for the MCVHD Application, Rate Limiter Application and Data Processing Application constitutes the complete IoT Application solution.

4.2 Containerized Microservices for the proposed IoT Solution

Containers are a lightweight approach to virtualization [54]. Containers are independent units which help in rapid development, testing, deployment and updating any IoT applications just about anywhere, i.e., in a cloud, within a local VM, or on physical hardware device. Using these advantages of container technology, this thesis chooses to use Docker as container Technology.

The microservices created for the proposed IoT solution, including Gateway data processing, VHD, and Rate Limiter, are containerized using the Docker technology. All Docker Containers for these microservices contain the required code and libraries for running the application. These containerized microservices now can be easily updated and deployed in any environment supporting Docker. Figure 4.2 shows Docker containerized microservices for the VHD application.

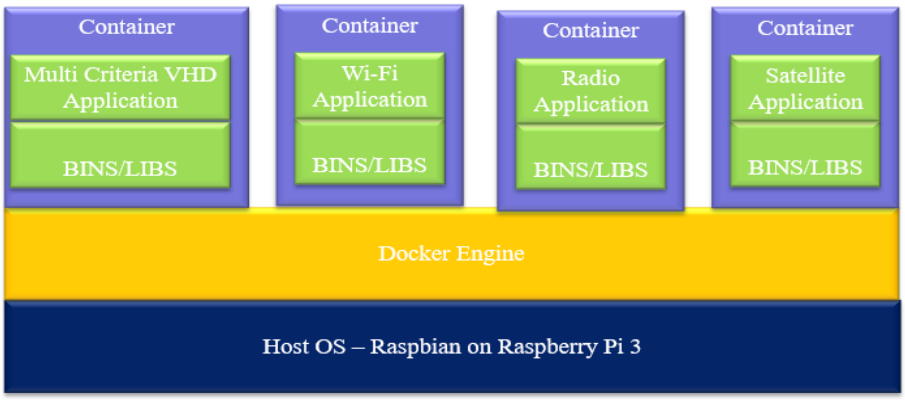

Figure 4.2 Docker containerized microservices for MCVHD application

In Figure 4.2, The containers are built for VHD, and data transmission for Wi-Fi, Radio and Satellite microservices. Each Docker container has its own library and runs independently. The Docker containers do not have to pre-allocate any RAM memory as they will take it as needed by the microservice. This characteristic of minimized wastage of resources makes Docker containers an ideal candidate for constrained IoT devices.

4.2.1 Containerizing Microservices using Docker Containers

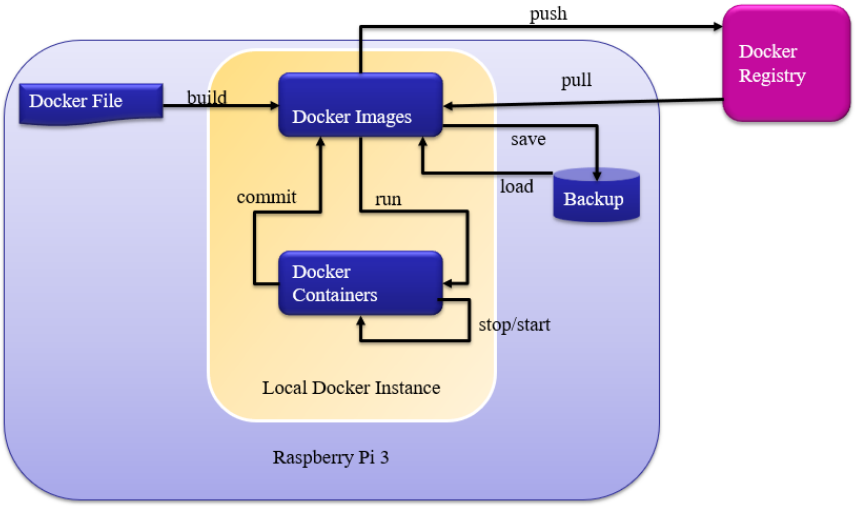

Figure 4.3 Docker components

Figure 4.3 presents the activities to build a Docker container and each step is described as follows:

1) Create a Docker file and write the code for the application
2) Build an image from the created Docker file. This image contains the entire project code with libraries and other dependencies.
3) The Docker Image can be now pushed to a Docker registry. Docker Registry is a cloud based repository to store Docker images.

4) An existing Docker image can be pulled from the Docker registry by others for development or testing or staging.

5) Using Docker *run* command create containers for the Docker images.

6) This Docker image could be saved in a backup data store

7) The running Docker containers can be stopped and restarted

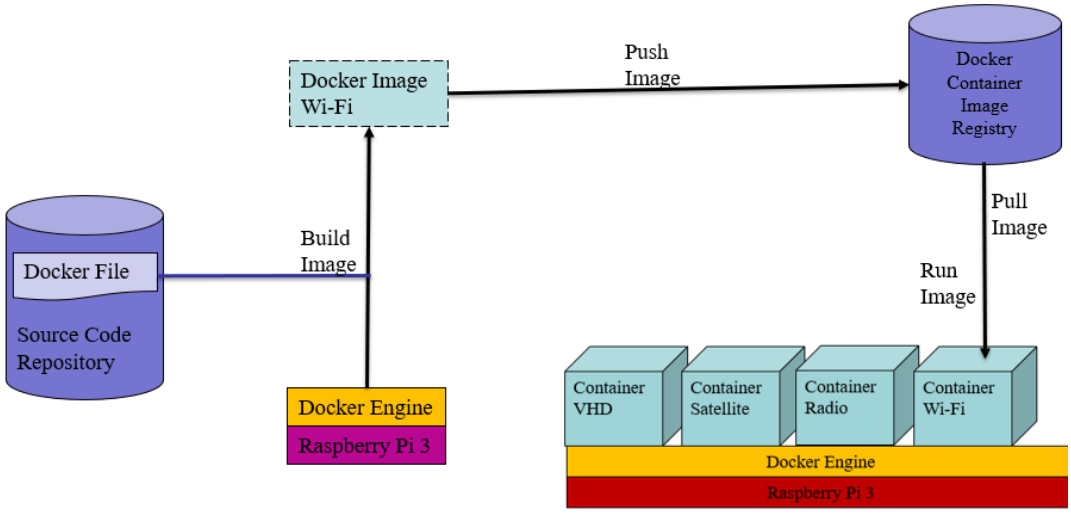

Figure 4.4 Activities of building Docker containers for the MCVHD microservices application in our proposed IoT solution

Figure 4.4 presents the activities of building Docker containers for the microservices of VHD application in our proposed IoT solution.

Docker files are text files that specify a list of instructions for building an image. Docker image is used to create and run the Docker containers. Six containers are created with individual Docker files containing corresponding configuration information, dependencies, libraries and shared binary files necessary to run application inside that container. The Dockerfile for MCVHD is presented in Figure 4.5. A Docker container is

created from Dockerfile for MCVHD and the container runs the Python based algorithm VerticalHandover.py with required Python libraries, dependences.

Dockerfile for MCVHD Algorithm running at IoT Gateway Node

```
# Pull Rasbian Debian OS base image for Raspberrry Pi
FROM resin/rpi-raspbian:jessie
MAINTAINER Amit Singh Gaur <amitgaur@cmail.carleton.ca>
# Install dependencies
RUN apt-get update && apt-get install -y \
   git-core \
   build-essential \
   gcc \
   python \
   python-dev \
   python-pip \
   python-virtualenv \
   python-pip cron\
   --no-install-recommends && \
   rm -rf /var/lib/apt/lists/*

RUN pip install pyserial simplejson configobj psutil gitpython
RUN apt-get -y install arduino-core
mkdir -p /home/pi/Handover

#Setting up users and permissions
RUN chown -R /home/pi/Handover
RUN find /home/pi/Handover -type f -exec chmod g+rwx {} \;
RUN find /home/pi/Handover -type d -exec chmod g+rwxs {} \;

#Cron Job for complete automation
COPY /home/pi/Handover -crontab /etc/cron.d/Handover
RUN chmod 775 /etc/cron.d/Handover
EXPOSE 80 443

# Start the cron daemon shell
COPY conFigure-and-start.sh conFigure-and-start.sh
RUN chmod +x conFigure-and-start.sh
CMD ./conFigure-and-start.sh

# Copy the requirements.txt first to leverage Docker cache
COPY ./requirements.txt /Handover/requirements.txt
WORKDIR /Handover
RUN pip install -r requirements.txt
VOLUME /Handover

# Copy the current directory contents into the container at Handover
ADD  .  / Handover
COPY . / Handover
ENTRYPOINT [ "python" ]

# Add the Python Handover.py to the Dockerfile
ADD  Handover.py /
# Run Handover.py when the container launches
CMD [ "python", "./Handover.py", "--port", "443"]
```

Figure 4.5 Dockerfile for MCVHD running at IoT Gateway Node

More descriptions for Docker files related to data communications using Wi-Fi, Radio, Satellite, Gateway node data processing, database process, visualization etc., are presented in Appendix A. Table 4.1 describes each command to create a Docker file, as presented in Figure 4.5.

Table 4.1 List of Commands used in Docker File for MCVHD

FROM: The base image for building a new image on top of the dockerfile.
MAINTAINER: Contains the name of the maintainer of the image
RUN: Execute a command during the build process of the docker image. Multiple RUN commands together using back slashes and && with a "no install recommend "on Debian/Linux images to install minimum dependencies. Removing repository cache files making image smaller with the help of **rm -rf** command.
ADD: Copy a file from the host machine to the new docker image.
COPY: ADD command will copy both entire directories and single files. COPY only copies files
ENV: Define an environment variable.
CMD: Used for executing commands when we build a new container from the docker image.
ENTRYPOINT: Default command that will be executed when the container is running.
WORKDIR: This is directive for CMD command to be executed.
USER: Set the user or UID for the container created with the image.
VOLUME: Enable access/linked directory between the container and the host machine.

Each Docker file contains corresponding set of instructions, dependencies and required libraries for running the python algorithm pointed to that Docker file. For the *Data Process* microservice the Dockerfile contains a Python based algorithm DataProcessing.py, PHP file, HTML, CSS file and required libraries and dependencies for filtering and aggregating the data coming from different sensor nodes into SQLite database and further visualization using Dashboard. Other Docker files contains the Python based algorithm for sending the data through Wi-Fi, Radio and Satellite based on the Handover algorithm.

4.2.2 Orchestration and management of microservices under multi-container environment with Docker Compose Architecture

Docker Compose is a Docker utility which makes containerization easier and effective by defining, creating microservice architecture, linking, and running multi-container applications. Additionally, Docker Compose provides many other features; for example, building images with a given Dockerfile, Scaling containers as a Service for container service, automate the containerization and full control on services running inside containers (Stop, Restart, Start and many other options). YAML files are used to configure application's services with the Docker Compose utility. After configuration of all services, the YAML file can be run which starts everything that needs to be done for application with all the linked services defined inside configuration file.

Docker Compose is basically a three-step process as described below:

1) First, define app's environment with a Dockerfile so it can be reproduced anywhere.
2) Second, Define all the services in Docker-compose.yml , that is required to run application So, services can be spin together in an isolated environment by running compose YAML file.
3) Lastly Compose will start and run entire application by running docker-compose up.

A multi-container deployment description YAML file is used which contains reference to all the microservices that are required to run the MCVHD application solution. The entire multi-container setup can be deployed and orchestrated for the MCVHD application by running the *Docker Compose up* CLI command with a YAML file which

will start the MCVHD application with all the linked services defined inside configuration file. The YAML code file defines the simplified and consolidated version of orchestrated multi-container microservices for proposed Containerized Vertical Handover Solution for IoT presented in Appendix B.

4.2.3 Configuration details of Docker Compose (YAML file) containing microservices inside containers

For the proposed multi-services containerized orchestration architecture for Vertical Handover, the thesis used the Docker Compose YAML file (Appendix B) to define the various services in the form of Docker images at different service level. The thesis has defined four services for the Vertical Handover mechanism and three services for handling the IoT data processing, storage and visualization through the dashboard. We used the *build* option to build an image from the Docker file for Vertical Handover service and tagged the image with the name "VHD" and deployed the image with reservation of memory and cpu so that the process can run effectively.

We have also used other options "links" and "Ports" to link, expose and map the running services externally with other containers so that the "VHD" container can communicate with other running container services for "Wi-Fi", "Radio" and "Satellite" and can effectively choose the mode of communication based on multi-criteria defined in Handover decision matrix. We have also defined the "Environment" and "volumes" option for each service to choose the relative required environment to run the service defined in the Docker file reference to that the volume location. The containers, Wi-Fi, Radio and

Satellite, use the same base image “VHD”, but configured to run differently and creates separate containers corresponding to their microservices. We tagged referenced images in service definition with the "image" option, and also use a "depends_on" to make sure the other containers are built and run based on referenced base image. Similarly, we have defined the microservices for IoT data processing node, IoT DB node and IoT Dashboard.

4.2.4 High-level Design Architecture of Orchestration of Containerized IoT Solution using Docker Compose (YAML file)

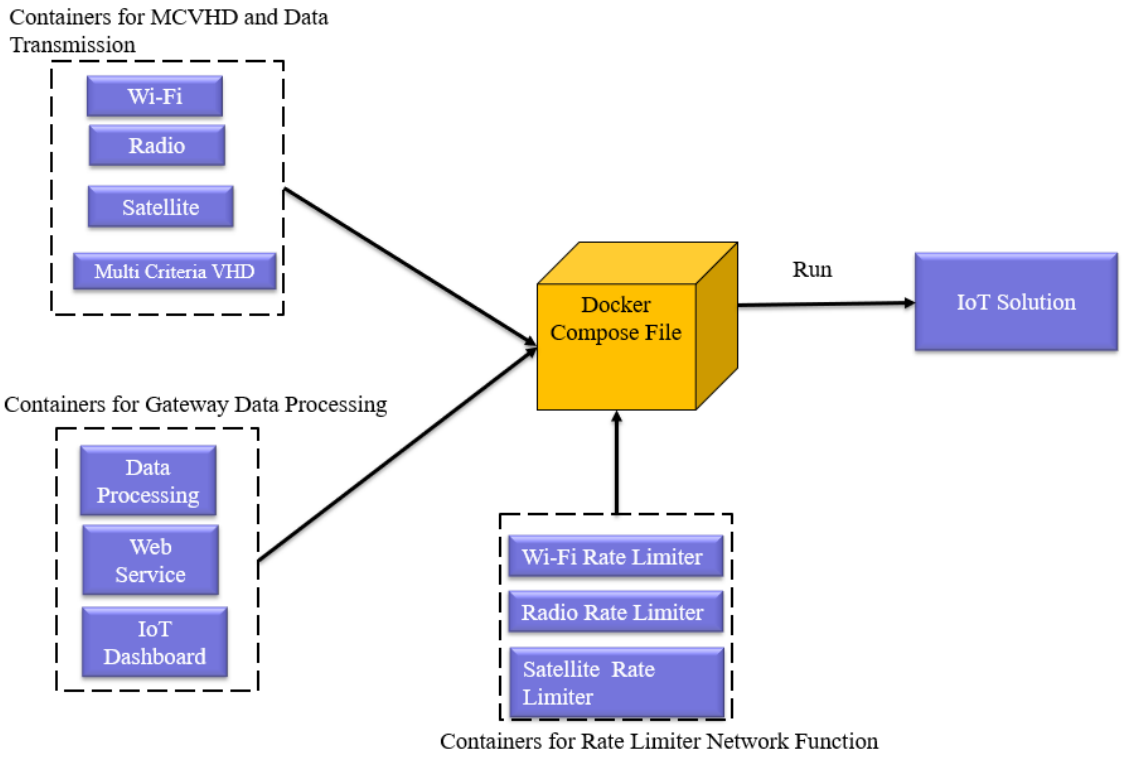

Figure 4.6 Containerized microservices using Docker Compose for the proposed IoT solution

The proposed IoT solution have three main categories of applications, MCVHD application, Rate Limiter and Gateway Data Processing. Each application is defined in the Docker container and these containers are linked together. In MCVHD, we have defined four isolated containers for their corresponding microservices like vertical handover decision, Wi-Fi, Radio and Satellite communications. Similarly, Rate Limiter application has three containers which runs microservices for data rate transmission through Wi-Fi, Radio and Satellite. Gateway Data Processing. There are three Docker containers, which are responsible for corresponding services such as Data Processing, Web Server and IoT Dashboard for visualization.

The proposed IoT solution has multiple applications which can be run together using Docker Compose. A single *docker-compose up* command can run multiple applications in various containers as shown in Figure 4.6.

4.3 Chapter Summary

The proposed IoT solution is designed as microservices architecture as it provides development of an application services in smaller individual functional components. Microservices architecture has advantages, e.g., each microservice is independent, hence easy to manage and there is a negligible effect on the entire application when any microservice fails.

Recently, the Container virtualization technology has become very popular since its lightweight and provide an ease of development, deployment and management of applications as individual and independent containers. The thesis made use of Docker container technology to containerize the microservices for the proposed IoT solution. For

performance analysis of the proposed containerized IoT solution, the experiments are conducted on a resource constrained IoT device like single board computers (Raspberry Pi 3) and the results are discussed in next chapter 5.

Chapter 5 Experimental Setup and Performance Evaluation

This chapter evaluates the performance of the proposed IoT solution using benchmark tools. One of the major parts of the proposed IoT solution is the design of an effective MCVHD method between Wi-Fi, Radio and Satellite networks for MTs. Section 5.1 discusses the experimental setup and the parameters used for performance evaluation of the proposed IoT solution. RSS is one of the important aspects of any network to be considered for handover. Traditional algorithms typically are also based on the RSS and RSS threshold value. In Section 5.2, we measure the RSS Quality of the three heterogenous networks, i.e., Wi-Fi, Radio and Satellite, received by the MT while moving away from the base station and roaming within the range of the network. Section 5.3 presents the performance evaluation of our proposed MCVHD algorithm in comparison with the traditional RSS Quality based algorithm. A set of experiments are conducted for network selection based on calculated priority ranking. The Wi-Fi network, the Radio network and the Satellite network are evaluated on the basis of the MCVHD algorithm and user weights (see Section 3.3.2.2) to determine the best available network. The best available network is given highest ranking.

Another important part of the thesis is dedicated to containerized microservices architecture of our proposed IoT solution. Section 5.4 presents an evaluation of the performance of Docker containers used for the VHD application, Data Processing application and the Rate Limiter application in comparison with the native environment, i.e., running the benchmark tools without including any virtualization layer. A brief summary of the chapter is presented in Section 5.5.

5.1 Experimental Setup

The experiments are conducted outdoor on a moving vehicle. The vehicle is equipped with the developed IoT system comprising of 2 sensors, an IoT Gateway with on board Wi-Fi, Radio Modem and Satellite Modem as shown in Table 5.1

Table 5.1 Hardware resources used in experimental setup

Resource	Hardware Device	Units
Sensor	JY901	2
IoT Gateway Device	Raspberry Pi 3 model B	1
Wi-Fi	Broadcom BCM43438 on board Wi-Fi Chipset in Raspberry Pi 3 Model B (IoT Gateway)	
	Wi-Fi Modem at Base Station	1
Radio Modem	XTend RF 900 MHz Modem (Transmitter) connected to IoT Gateway	1
	XTend RF 900 MHz Modem (Receiver) at Base Station	1
Satellite Modem	RockBLOCK Mk2 (Iridium 9602 SBD) with onboard antenna connected to IoT Gateway	1

The sensors are connected serially to the Raspberry Pi 3 which acts as the IoT Gateway. The IoT Gateway runs three applications: Firstly, data processing in which it filters, aggregates data received from sensors, and stores into the SQLite database for local visualization using Apache2 web server. Secondly, it performs vertical handover decision based on the multi-criteria method to handover between the Wi-Fi, the Radio and the Satellite networks. Thirdly, it performs data transmissions according to the data rate limiter network function. For transmitting sensor data, the IoT Gateway is serially connected to the Radio Modem (transmitter) and the Satellite Modem using USB to Serial connection as shown in Figure 5.1.

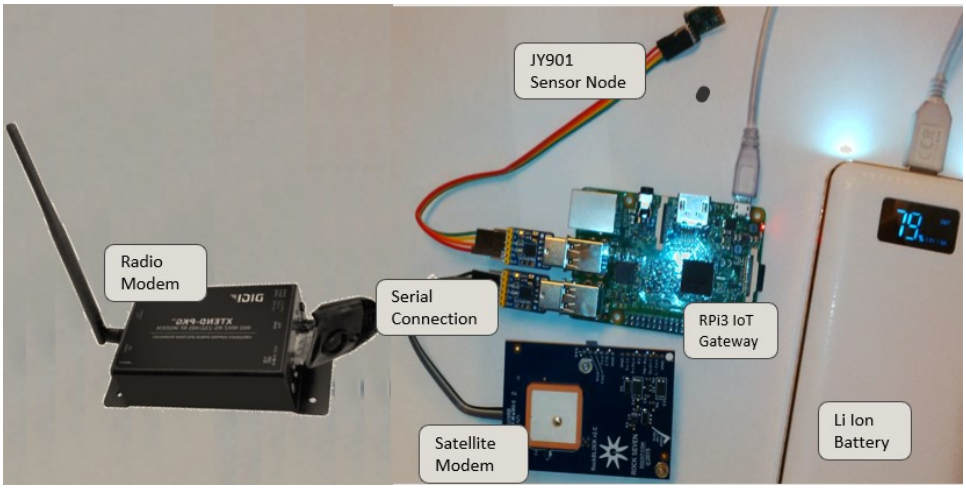

Figure 5.1 Experimental Setup for IoT MT

Figure 5.2 presents experimental setup for IoT MT and Base Station with Wi-Fi, Radio and Satellite Communications.

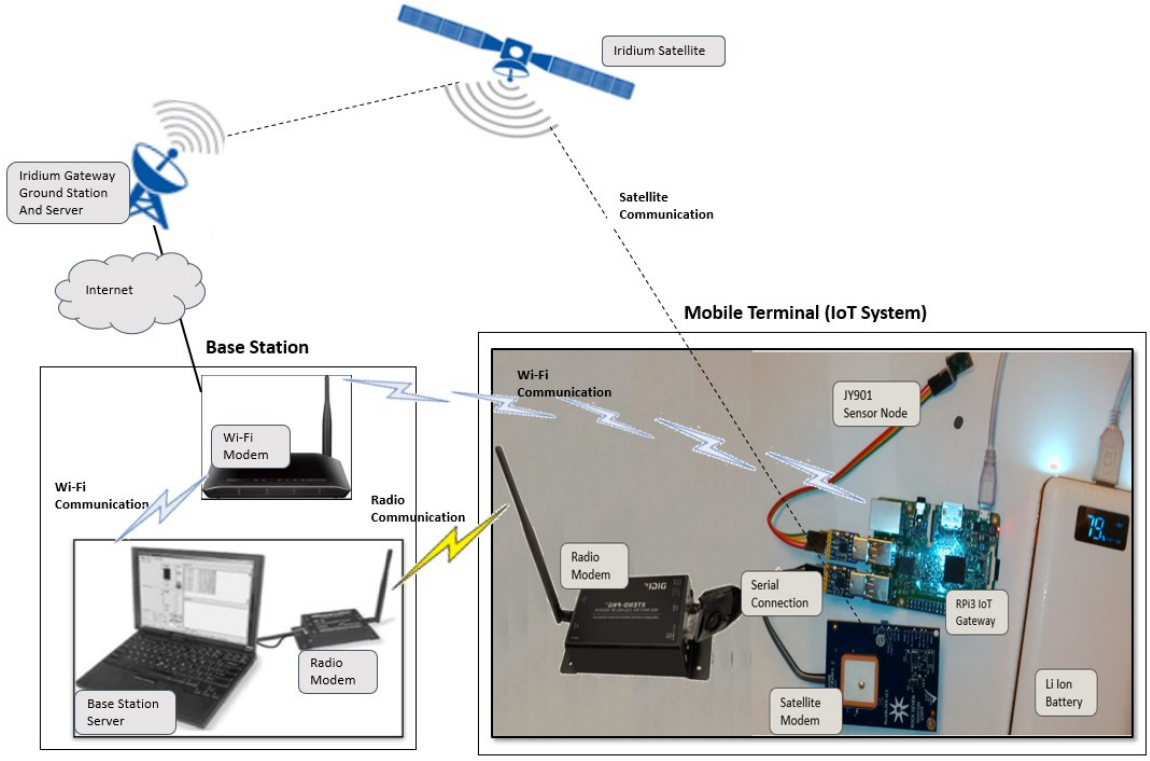

Figure 5.2 Experimental Setup for IoT MT and Base Station with Wi-Fi, Radio and Satellite Communications

The Satellite Modem transmits data to Satellite Iridium Server via Iridium Satellite Constellation. The Satellite Iridium Server sends data to the base station server. The base station is comprised of a Wi-Fi Modem and a Radio Modem as a receiver which are connected to a Server (Ubuntu 14.04). The base station server collects data received from the Wi-Fi, Radio and Satellite Iridium Server, then it sends the data to the cloud via the Internet. A dashboard cloud application is developed for visualization using freeboard.io [55].

Table 5.2 exhibits the parameters and their permissible value range used for our proposed MCVHD algorithm. Handover multi-criteria considered here are RSS Quality, data service cost, data rate, network latency, reliability, network coverage, power requirement, mobile terminal velocity which are denoted as P1, P2, P3, P4, P5, P6, P7, and P8, respectively.

Table 5.2 Parameters for MCVHD algorithm

Priority Matrix Parameters	Parameter Notation	Available Heterogeneous Networks value range		
		Wi-Fi	Radio	Satellite
RSS Quality (dBm converted into Percentage)	P1	0-100 (%)	0-100 (%)	0-100 (%)
Data Service Cost (per 100Kb) in CAD	P2	0.005 – 0.010	No Data Cost	215 (RockBlock Service) Depends on Subscriptions
Data Rate (Mbps)	P3	30 – 50	0.0096 – 0.0192	0.0096 – 0.0192
Network Latency	P4	Low	Medium	High

Reliability	P5	Low	Medium	High
Network Coverage (km)	P6	0.05 (indoor) - 0.1 (outdoor)	Up to 5	Very Large (> 1000)
Power Requirement	P7	80 mA-115 mA, Constant 5V Voltage, Supplied by Raspberry Pi 3	90 mA – 180 mA with Supply Voltage (7 - 18V)	100 mA supplied at constant 5V power source
Mobile Terminal (Vehicle) Velocity (km/h)	P8	20 – 60	20 – 60	20 – 60

The user defined weights are presented in Table 5.3 and the linguistic terms are converted to numeric values using the fuzzy logic conversion scale (see Section 3.3.2.2). User defined weights are converted to numerical values and are normalized so that the sum is equal to 1.

Table 5.3 User defined weights for parameters used in MCVHD algorithm

Priority Matrix Parameters	**Weight Notation**	**User defined weights linguistic value**
RSS Quality (dBm converted into Percentage)	W1	High
Data Service Cost (per 100Kb) in CAD	W2	Low
Data Rate (MBPS)	W3	Medium
Network Latency	W4	Medium
Reliability	W5	High

Network Coverage (km)	W6	Medium
Power Requirement	W7	Low
Mobile Terminal Velocity (km/h)	W8	Medium

5.2 RSS Quality measure for Wi-Fi, Radio and Satellite network

The following subsections present the experimental results of the network performance in terms of RSS Quality measured for the Wi-Fi, Radio and Satellite networks. This experiment has been conducted outdoor by moving within a range of 40 meters from the base station. The reason to choose 40 meters of range is to evaluate simultaneously the RSS Quality of the Wi-Fi, Radio and Satellite networks within the network range. As per the experiments conducted, though the Radio and Satellite network coverage is available beyond 40 meters, but the RSS Quality of the Wi-Fi tends to decline after 40 meters. The reason for declining RSS Quality of Wi-Fi is due to more buildings outdoor and low power antenna of the used Wi-Fi Modem. A RSS Quality value of 50% has been taken as the threshold for all the three networks. All the experiments are conducted for 2 minutes. A detailed analysis of the results obtained is presented in the below subsections.

5.2.1 Wi-Fi RSS Quality Measurement

The RSS Quality of the Wi-Fi network is measured between Wi-Fi of the IoT Gateway and the ISP Rogers Wi-Fi router providing the access point at the base station. A command line utility, *iwconfig* is used at the IoT Gateway to measure the RSS Quality of the Wi-Fi network. Figure 5.3 shows the result of the measured RSS Quality of the Wi-Fi network

during a period of 2 minutes when moving away from the base station and roaming within a range of 40 meters.

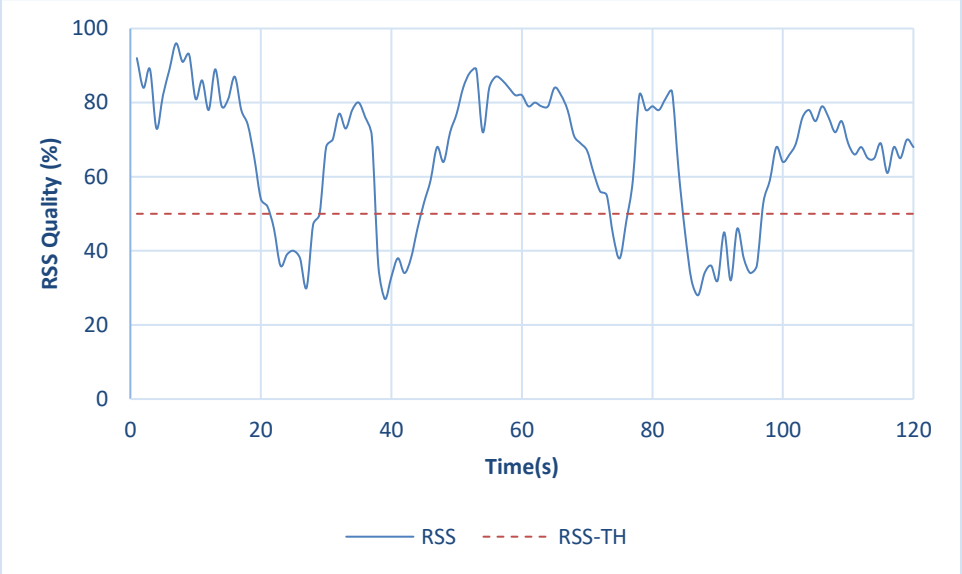

Figure 5.3 RSS Quality of Wi-Fi network

As per the result shown in Figure 5.3, the RSS Quality for the Wi-Fi network shows more fluctuations. The maximum RSS Quality is observed around 95% when the vehicle is near the Wi-Fi base station, while the RSS Quality drops to around 28% when the vehicle moves away from the base station. The mean value of RSS Quality is observed to be around 72%, which is considered fairly good network RSS Quality. It can be also observed that the RSS Quality drops several times below the threshold limit, which is 50% of the RSS Quality but again receives high RSS Quality when the signals are available. This signal strength variation in the Wi-Fi network could possibly affected by many factors [56], including:

1) The nature of physical obstructions and/or radio interference in the surrounding area
2) Distance between the device and other Wi-Fi communication endpoints

3) The strength of the device's Wi-Fi radio transmitter/receiver

4) ISP and type of Wi-Fi Modem

5) Number of wireless connections present at Wi-Fi Modem

This experiment is conducted 5 times and during each test the results shows similar pattern of high fluctuations in the RSS Quality. Hence, we can conclude that the nature of the signal quality of the Wi-Fi network is more fluctuating and this could possibly affect the vertical handover decision.

5.2.2 Radio RSS Quality Measure

The RSS Quality of the Radio network is measured between the XTend Radio Transmitter Modem at MT transmitting data at 900 MHz to the XTend Radio Receiver Modem at the base station as shown in Figure 5.2. "*AT RS*" is an *AT* command used at the IoT Gateway to measure the RSS Quality of the Radio network.

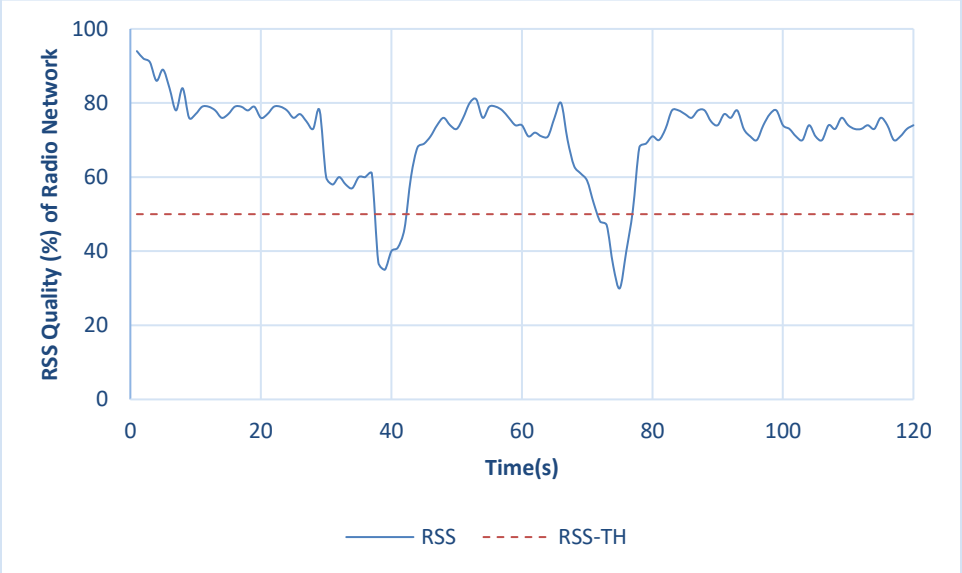

Figure 5.4 RSS Quality of Radio network

Figure 5.4 shows the variation of the RSS Quality for the Radio network experienced by the IoT Gateway as functions of time. The observation is taken for around 2 minutes within the range of 40 meters. The RSS Quality of the Radio network shows a less fluctuating behavior as compared to that of the Wi-Fi network (see Figure 5.3) The maximum RSS Quality is observed around 96% when the vehicle is close to the base station, which is similar to the case for the Wi-Fi network. It can be also observed that the RSS Quality drops to around 31% when the vehicle moves away from the Radio base station. The mean value of the RSS Quality is observed around 70% which is close to the average value of the RSS Quality in the Wi-Fi network.

This experiment is conducted 5 times and the results shows similar pattern in each attempt. With this experiment we can conclude that the highest and the mean RSS Quality value of Radio network is similar to that of the Wi-Fi network and these similar values could make Wi-Fi and Radio as competitive candidates when the vehicle is close to the base station. Though the similarity exists in highest and mean RSS Quality values, it is worth noticing that the Wi-Fi network fluctuates more often as compared to the radio network, which could affect the decision for vertical handover.

5.2.3 Satellite RSS Quality Measure

The RSS Quality of Satellite network is measured using the *AT* command "*CSQ*" at the IoT Gateway. The signal strength is measured between the satellite Modem (transmitter) at the IoT Gateway MT and the Iridium Satellite constellation as shown in Figure 5.2. The antenna on the Satellite Modem must be positioned to capture as much of the sky as possible. Since Iridium satellites are not geostationary, they pass over the site at different

longitudes. During one pass the Satellite Modem antenna may have a perfect view of the satellites, whereas during another pass, it may not be as good (if there are obstructions). This could result in some loss of data. For best signal reception, the Satellite Modem antenna is mounted on the top of the vehicle so that it has an unobstructed view of the sky.

The experiment is conducted 5 times and it shows the similar constant pattern as shown in Figure 5.5. The RSS Quality of the Satellite network is observed to be least fluctuating and mainly varies between 70 to 73%. It is interesting to note that the RSS Quality dropped once below the threshold limit down to 28%. The reason for this drop in the RSS Quality value is that during that time the vehicle passed through a densely planted area which restricted the clear view of the sky. During such a low RSS Quality, i.e., below 40% the Satellite Modem does not transmit data and stores the data in the buffer to be transmitted once the RSS Quality is achieved more than 40%.

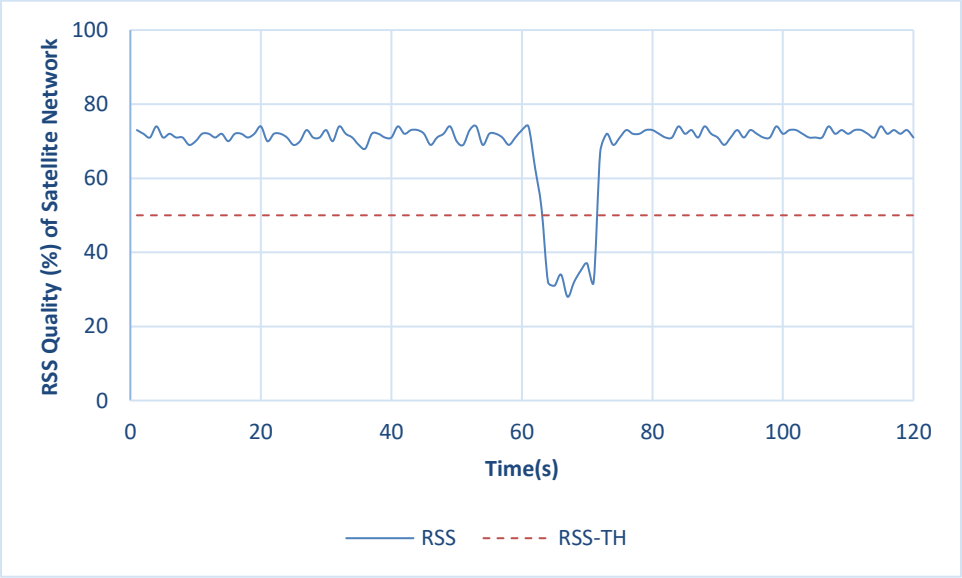

Figure 5.5 RSS Quality of Satellite network

We can conclude that though the RSS Quality of the Satellite is lower than the Wi-Fi network and Radio network, it is least fluctuating and remains close to constant for a wide network coverage.

5.3 Performance Evaluation of the Proposed MCVHD versus Conventional RSS Quality based VHD

The following subsections presents the experimental results of the proposed MCVHD algorithm and the conventional RSS Quality based VHD algorithm. The experiments show the result as an average value of 10 experiments conducted to evaluate the performance of both of the algorithms. The experiments are conducted, and results are evaluated for scenarios when the vehicle is close to the base station, i.e., highest probability of availability of all three Wi-Fi, Radio, Satellite networks, and when the vehicle is moving away from the base station where the network declines in order of Wi-Fi then Radio.

The performance of the proposed MCVHD algorithm is evaluated on the basis of four different experiments. Firstly, on the basis of the network selected in different scenarios by calculating the priority ranking of the network. Secondly, the performance evaluation on the basis of the ratio of unnecessary handovers occurred during a period of time. Thirdly, the ratio of Vertical Handover failure is calculated for both of the aforementioned methods for performance evaluation. Finally, an experiment is conducted whenever a vertical handover occurs between the Wi-Fi, Radio and Satellite networks to calculate the handover time. A detailed result analysis is presented in the following subsections.

5.3.1 Priority Ranking

Priority Ranking is used for selecting best network among the available heterogeneous networks based on multi-criteria and their associated weights using the MCVHD algorithm. The Priority Ranking is according to the descending order of the relative closeness value of each network. Hence, the network with the highest value of relative closeness is selected as an optimal network for handoff in the available heterogeneous networks. Traditional RSS Quality based VHD method only considers the best RSS Quality network, whereas the proposed MCVHD method evaluates the best network based on certain network parameters and given weights, as shown in Table 5.2 and Table 5.3.

5.3.1.1 Network selection priority ranking when MT is close to base station

This experiment is conducted by running both the candidate algorithms for VHD when the vehicle is close to the base station.

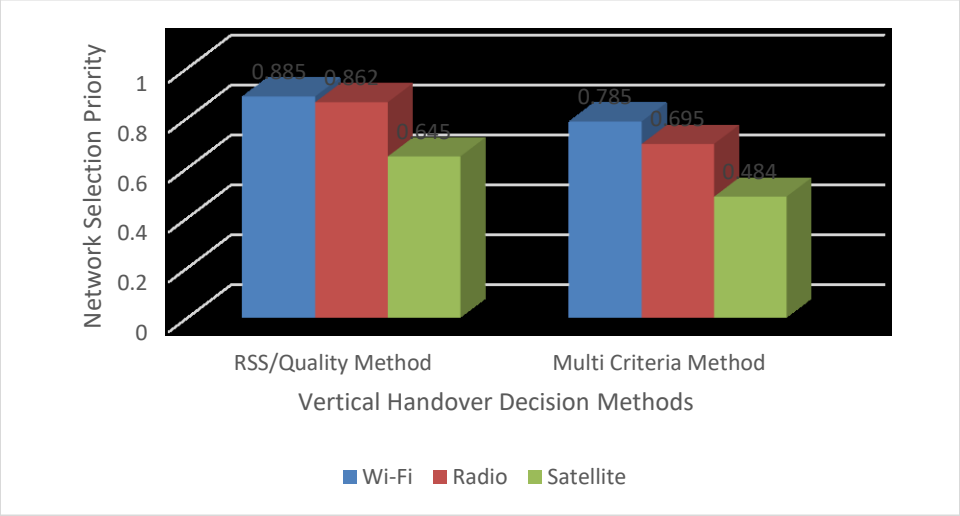

Figure 5.6 Network Selection Priority Ranking when MT is close to base station

Figure 5.6 shows the network selection priority ranking for Wi-Fi, Radio and Satellite networks using the conventional RSS Quality based VHD Method and MCVHD Method.

It can be observed that by using the conventional RSS Quality based VHD method the ranking of Wi-Fi is 0.885, which is highest, but it has another competitive candidate, i.e., the Radio network which shows a very close figure of 0.862. This negligible closeness between the Wi-Fi and the Radio networks makes both as an ideal candidate for selecting the best network and thus can contribute to more vertical handovers due to fluctuating nature of Wi-Fi.

While using the proposed MCVHD method the ranking of Wi-Fi network is the highest followed by the Radio and then the Satellite networks. Although both the algorithms generate similar results when the vehicle is close to base station where all the three networks are available in the best capacity, a quite difference can be observed between the Wi-Fi and Radio network selection. This is because in the MCVHD algorithm, different criteria that are used to select the best network and RSS Quality only plays 20% of the weightage. The other dominant criteria like data rate is always high for Wi-Fi which makes Wi-Fi quite different from the Radio network, even though Wi-Fi drops a little below the Radio network RSS Quality.

It can be also observed that the ranking of Satellite network is always low in both methods mainly due to higher cost of data transmissions.

5.3.1.2 Network selection priority ranking when MT is moving away from the base station

This experiment is conducted by running both the conventional and the proposed algorithms for VHD when the vehicle is moving away from the base station. Figure 5.7

shows the network selection priority ranking for the Wi-Fi, Radio and Satellite networks using the conventional RSS Quality based VHD Method and the MCVHD Method. The priority ranking of Wi-Fi is noticeably lower than that of Radio and Satellite networks, since the range of Wi-Fi network is very low, i.e., 30-50 meters. As the vehicle moves away from the base station the Wi-Fi RSS Quality decreases significantly.

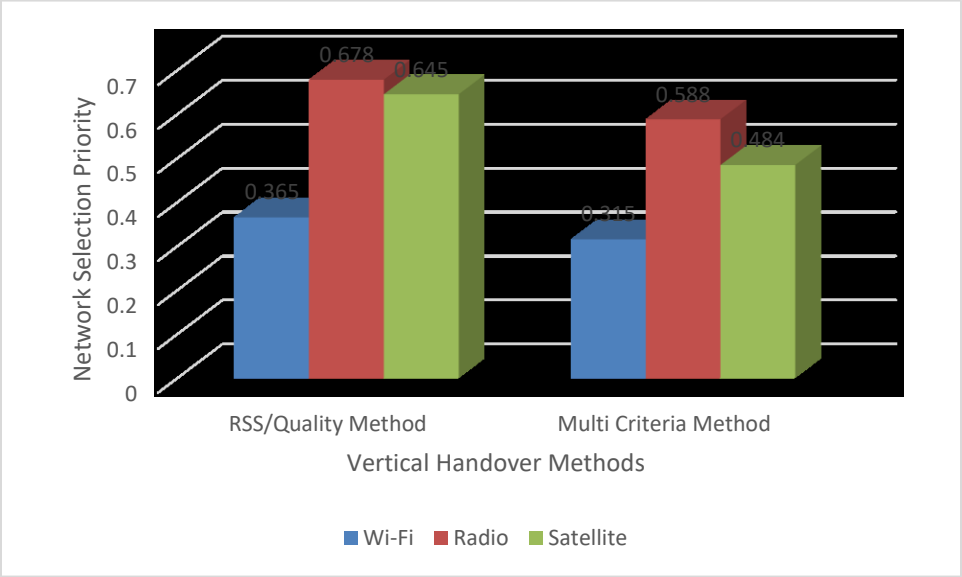

Figure 5.7 Calculated Network Selection Priority Ranking when MT is moving away from the base station

The results show that the priority ranking of Radio is higher in both the algorithms, but it must be noticed that ranking values of Radio and Satellite networks are quite close in the conventional RSS Quality based VHD method. Any fluctuations in the Radio network could cause unnecessary handover to the Satellite network. While the MCVHD algorithm considers multi-criteria in which cost of data transmission is also critical along with the RSS Quality. Hence, the Radio shows a noticeably higher-ranking value as

compared with the Satellite Network. Even though the Radio network RSS Quality could fluctuate, but it will still select Radio network unless the RSS Quality drops very low as in case of crossing the range of Radio Network.

5.3.1.3 Network selection priority ranking when MT is very far from the base station

This experiment is conducted by running both the algorithms for VHD when vehicle is very far, at around 6 km from the base station. Figure 5.8 shows the network selection priority ranking for the Wi-Fi, Radio and Satellite networks. It can be observed that the ranking of the Wi-Fi network is very low as compared to the Radio and Satellite networks. The reason for a low ranking is due to no availability or limited availability of Wi-Fi network as the vehicle is very far from the base station.

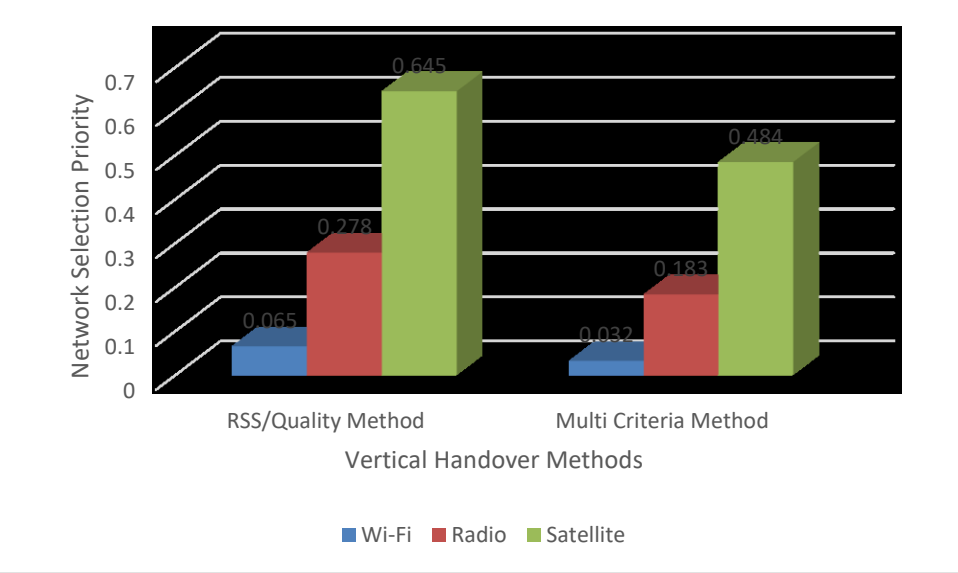

Figure 5.8 Calculated Network Selection Priority Ranking when MT is at around 6 km far from the base station

The ranking value of Radio network is also quite low as compared to that of the Satellite network, since the vehicle is moving out of the range of Radio network which is typically up to 5 km. The Satellite network shows the highest priority ranking for selection even though the cost of data transmission is very high, since it has a higher network range of more than 1000 km and is only available network after the Radio.

5.3.2 Ratio of unnecessary Vertical Handover

Reducing the number of unnecessary Vertical Handovers is usually preferred for constrained IoT devices as frequent unnecessary handovers would cause wastage of network resources, processing overhead, wasted of power consumptions, costs, etc. Ratio of unnecessary Vertical Handover can be defined as the ratio of the number of unnecessary Vertical Handover to the total number of Vertical Handover occurred during a period of time.

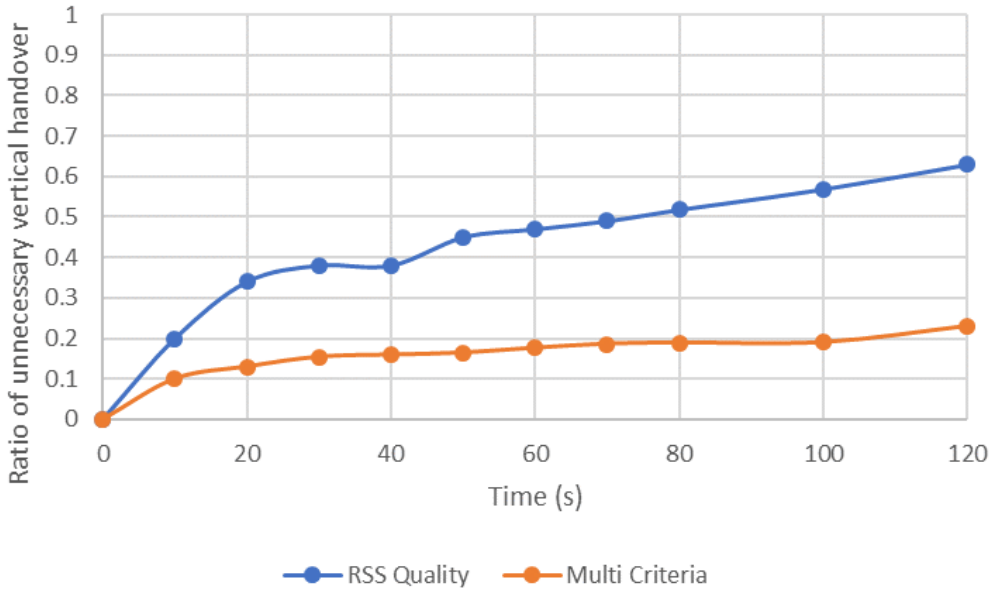

Figure 5.9 Ratio of unnecessary vertical handover when MT velocity is 20 km/h

This experiment has been conducted with vehicle velocity of 20km/h moving away from the base station. It can be observed from Figure 5.9 that the ratio of unnecessary handovers is higher in the conventional RSS Quality based VHD method in comparison to the proposed MCVHD method. The reason for more unnecessary Vertical handover in the conventional RSS Quality based VHD method is because it considers a network to be the best only based on highest RSS Quality among all the available networks above the threshold value. It is observed that the RSS Quality of Wi-Fi is more fluctuating, as shown in Figure 5.3, and in that case the probability of unnecessary Vertical Handover to Radio or Satellite network becomes high.

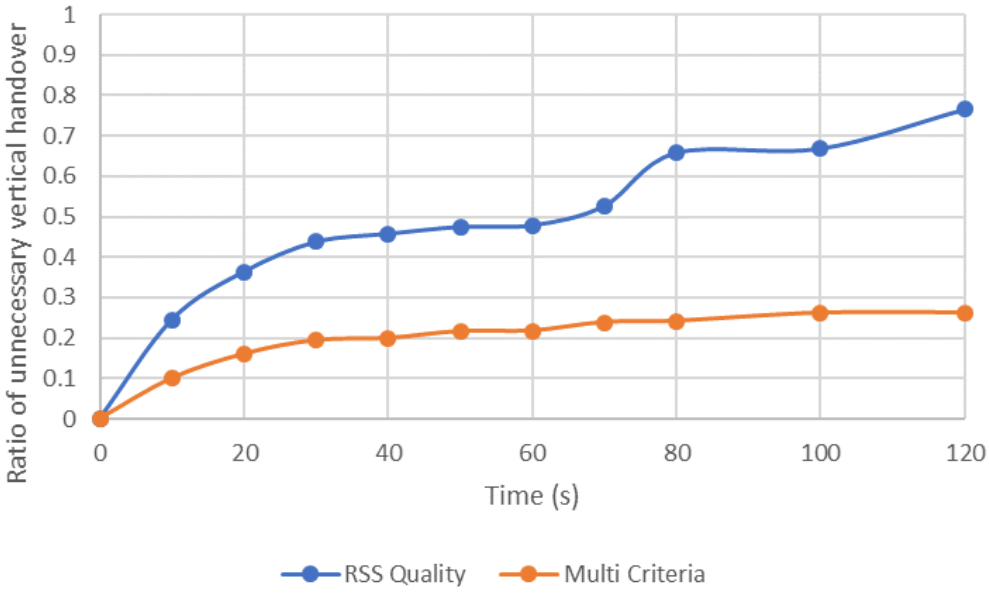

Figure 5.10 Ratio of unnecessary vertical handover when MT velocity is 60 km/h

Since the proposed MCVHD algorithm is designed to minimize the unnecessary handover instances, even though when the velocity of the vehicle increases, the results are almost same. As illustrated in Figure 5.10, when the velocity is increased to 60km/h, the proposed

MCVHD algorithm yields much lower ratio of unnecessary handover instances than the RSS Quality based VHD method as time goes by.

5.3.3 Ratio of Handover Failures

A Handover Failure occurs when the Handover request is initiated but the handover is not completed successfully. There could be several reasons for Handover Failures such as unavailability of sufficient resources to complete the handover to the target network or MT moves away from the coverage range of the targeted network before completion of handover process.

The ratio of Vertical Handover Failure can be defined as the ratio of the number of failure during Vertical Handover process to the total number of Vertical Handover occurred during a period of time. Figure 5.11 shows that the MCVHD algorithm outperforms the the conventional RSS Quality based VHD algorithm by minimizing the failure in the Vertical Handover process.

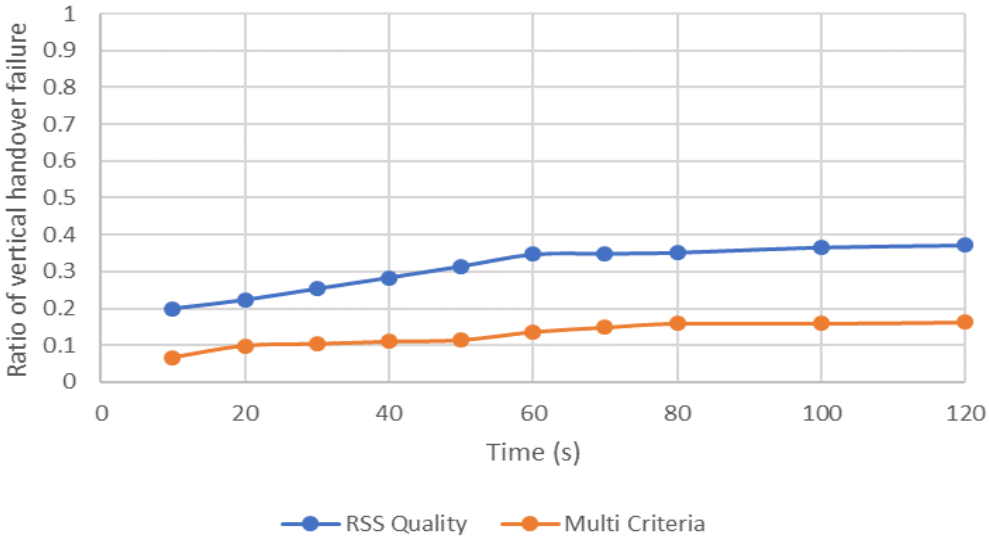

Figure 5.11 Ratio of vertical handover failure when MT velocity is 20 km/h

The ratio of vertical handover failure in the conventional RSS Quality based VHD increases from 0.2 (20%) to 3.8 (38%) whereas the proposed MCVHD shows minimized failure ratio between 0.08 (8%) to 1.6 (16%). Dependency on the RSS Quality and fluctuating nature of the Wi-Fi network are the main reasons for the poor performance of the conventional RSS Quality based VHD algorithm in terms of vertical handover failure. As the RSS Quality of Wi-Fi drops, the conventional RSS Quality based VHD algorithm initiates the process of Vertical Handover to next best network with higher RSS Quality. Before even the Vertical Handover process completes, the Wi-Fi network receives again a higher RSS Quality, which results in Vertical Handover Failure.

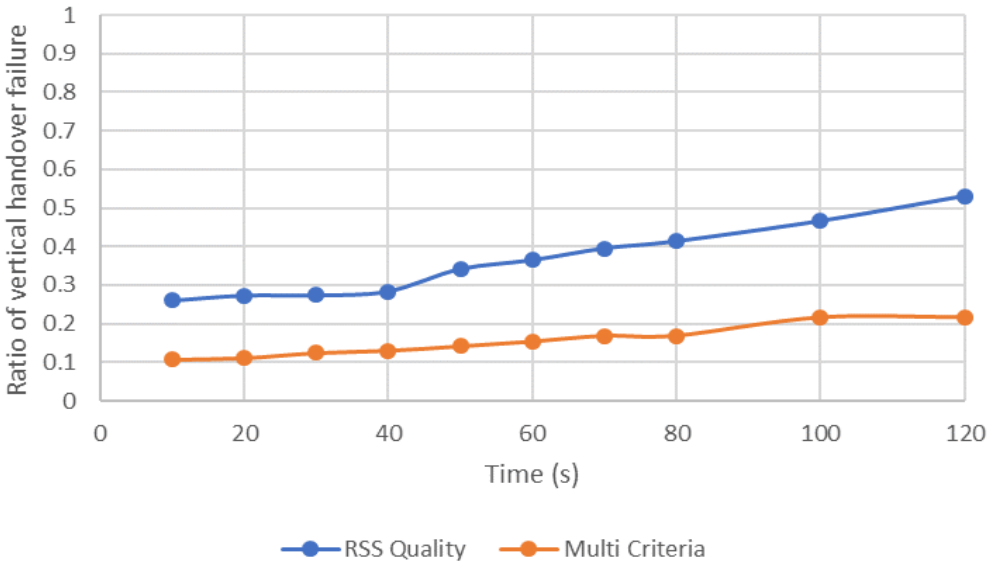

Figure 5.12 Ratio of vertical handover failure when MT velocity is 60 km/h

The similar experiment is also conducted with increased velocity of the vehicle. As shown in Figure 5.12, when the velocity is increased to 60km/h, the proposed MCVHD algorithm yields lower ratio of handover failure as compared to the RSS Quality based VHD method. It can be concluded that the proposed MCVHD outperforms the traditional

RSS Quality based VHD method and has a stable behavior even by changing the velocity of the Vehicle.

5.3.4 Handover Time

The duration between initiation and completion time of the handover process is referred to as handover time. The initiation corresponds to the instant when the VHD algorithm selects the highest ranked network and successfully connects to it. The completion time refers to the time taken by the first acknowledgement packet to be received by the IoT Gateway after handover.

5.3.4.1 Handover Time Between Wi-Fi and Radio

The handover time is measured by the IoT Gateway on MT and an average of 20 different runs are taken for the experiment. The handover time values are calculated between the Wi-Fi and Radio network transitions. In Figure 5.13, the instantaneous values of the handover time are shown as functions of the considered transitions.

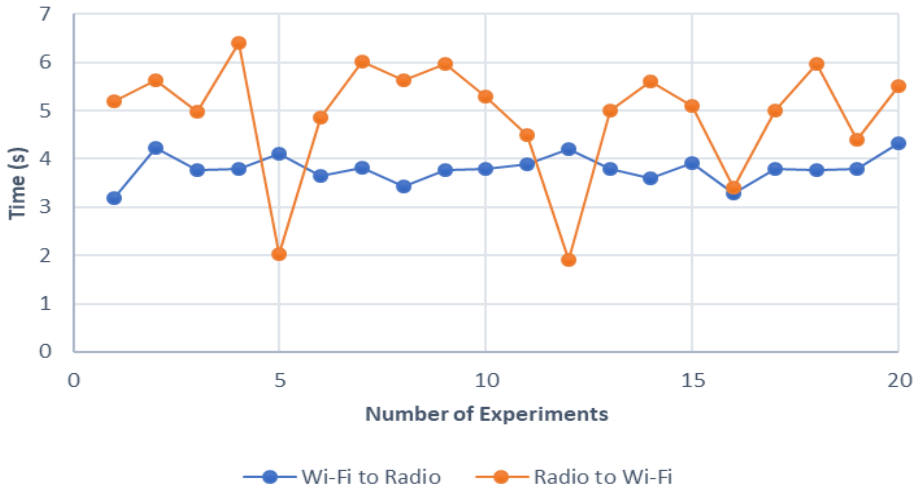

Figure 5.13 Handover Time between Wi-Fi and Radio

In Table 5.4, the average handover time is shown with corresponding standard deviation and confidence interval for the Wi-Fi to Radio and the Radio to W-Fi transitions, respectively.

Table 5.4 Results of Handover Time between Wi-Fi and Radio

	Mean	Standard Deviation	Confidence Interval (95%)
Wi-Fi to Radio	3.89	0.248	3.89 ± 0.12
Radio to Wi-Fi	4.76	1.302	4.76 ± 0.54

Figure 5.13 exhibits that the Wi-Fi and Radio networks have very different behaviors. In particular, the handover from Wi-Fi to the Radio network exhibits an almost constant handover time, around its average of 3.89 s (Table 5.4). However, there are few samples also in the region between 4 s and 5 s. On the other hand, the transition from the Radio network to Wi-Fi network shows lower performance than the previous scenario, since it presents a higher average value of 4.76 s (Table 5.4) and a much greater standard deviation.

At the same time, it can be also noticed that the minimum value of handover time to Wi-Fi is very small, which is around 1.8 s. The reason for a lower handover time is because usually MT logouts from a given network once the VHD application selects the other network for handover to save energy. Sometimes the logout fails, and the remote authentication server keeps the authentication state for a certain timeout before to automatically logout the user. In this case, the Radio to Wi-Fi transitions can experience a small handover time, since the MT is already authenticated to the network.

5.3.4.2 Handover Time Between Radio and Satellite

Similar experiment is conducted on MT to measure the handover time between the Radio and the Satellite networks. This experiment is conducted in 20 runs and the result is shown in Figure 5.14 and statistical information of the experiment in terms of standard deviation and confidence interval is presented in Table 5.5.

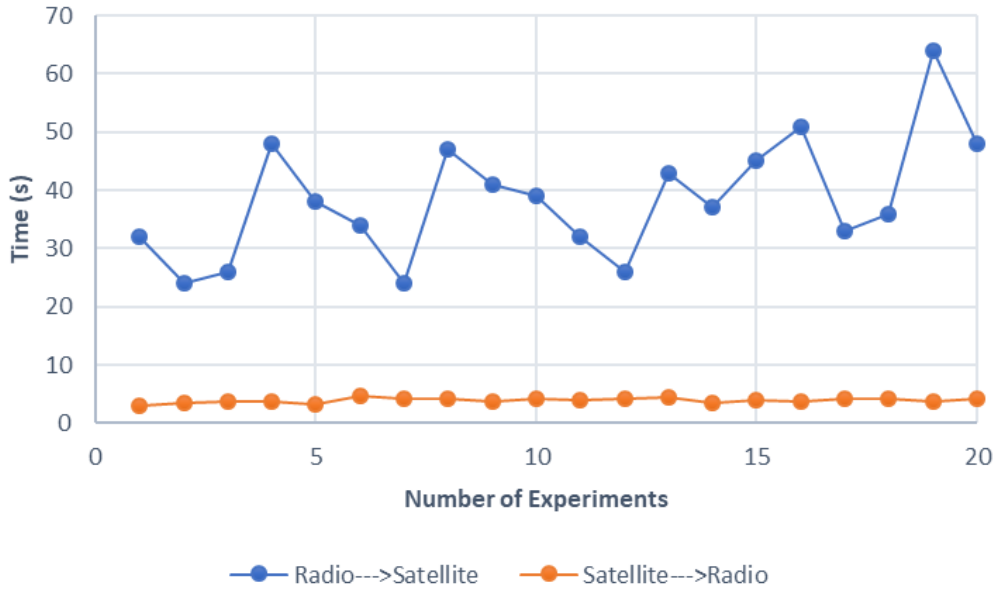

Figure 5.14 Handover Time between Radio and Satellite

Table 5.5 Results of Handover Time between Radio and Satellite

	Mean	Standard Deviation	Confidence Interval (95%)
Radio to Satellite	38.33	10.325	38.33 ± 4.55
Satellite to Radio	3.91	0.386	3.91 ± 0.17

The handover from Radio to Satellite shows a very high handover time in the range of 26 s to 64 s as shown in Figure 5.14. The higher handover time accounts for activating the satellite Modem to send data and latency to receive acknowledgement back of the first

data packet sent. The average time for handover to Satellite network is recorded as 38.33 s with high standard deviation of 10.325 s, presented in Table 5.5. While the handover time from Satellite to Radio shows a similar pattern as that of Wi-Fi to Radio network transmission of around 3.9 s.

The proposed MCVHD algorithm is designed to select Satellite network only when other network are the least available since Satellite network has high latency and high service cost.

5.4 Performance evaluation of Docker container environment

The following subsections discuss the experimental results and performance of our proposed containerized IoT solution by using benchmark tools. We compare and evaluate the performance of the Docker container technology on the IoT Gateway device, which is a resource constrained Raspberry Pi 3 in our proposed IoT system, with native performance. Native performance refers to running benchmark tools directly on top of the operating system without any Docker containers (virtualization layer). We used the Docker version 1.11.1 and Raspbian Jessie as the operating system with Linux Kernel Version 4.9. Table 5.6 shows the system configuration for both of the experimental setup, i.e., native and the Docker container environments.

Table 5.6 System configuration of Native and Containerized environment

	Native Raspberry Pi 3	Raspberry Pi 3 with Docker
Processor Chipset	Broad BCM 2837 64 Bit Quad Core ARM Cortex A53 at 1.2 GHz	Broad BCM 2837 64 Bit Quad Core ARM Cortex A53 at 1.2 GHz
RAM	1GB SD RAM	1GB SD RAM
Storage	MicroSD 32 GB	MicroSD 32 GB

Operating System	Native OS (Raspbian Jessie with Linux Kernel Version 4.9)	Native OS (Raspbian Jessie with Linux Kernel Version 4.9) + Docker version 1.11.1

5.4.1 Synthetic Benchmark Performance

Synthetic benchmarks tools are used to evaluate the performance of the proposed containerized IoT solution. These benchmark tools allow to generate various types of workloads to challenge a specific hardware segment of Raspberry Pi 3 which acts as the IoT Gateway. In order to evaluate the performance and impact of the presence of a virtualization layer, CPU, Memory I/O and Disk I/O are tested. The power consumption of the Raspberry Pi 3 during the execution of each single test is also reported. This is measured by using the Linux command utility *powertop*[57]**.** The performance of both the Native and the Docker container environments have been evaluated by running benchmark tools .

5.4.1.1 CPU Performance

To evaluate the CPU performance, the *sysbench* 0.4.12 benchmark tool is used. *sysbench* is a modular, cross-platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system running a database under intensive loads [58]. *sysbench* measures execution time (measured in seconds) for all processed requests to display statistical information. A lower execution time implies a better performance. Experiments are conducted to measure average execution time of the proposed IoT solution and Figure 5.15 shows the result of CPU performance for an average of 20 tests.

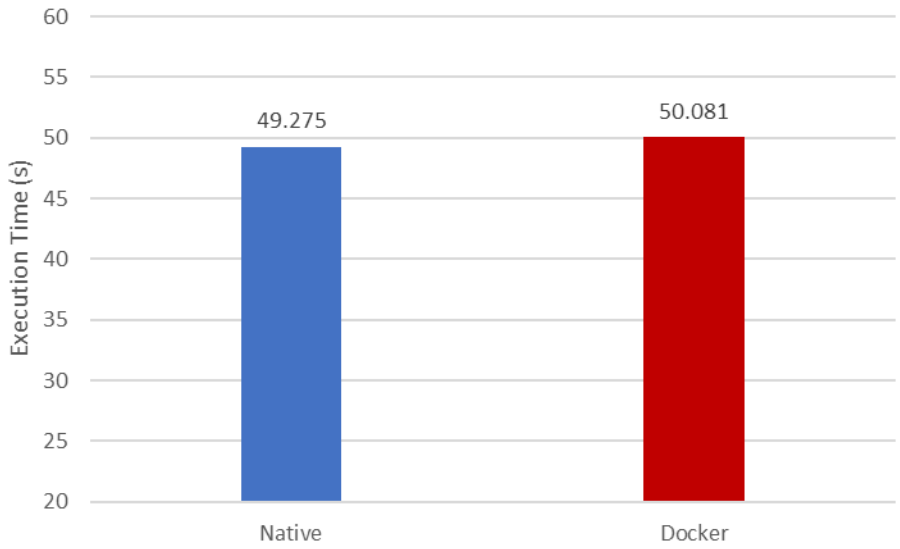

Figure 5.15 CPU Performance

Table 5.7 Results of CPU Performance (seconds)

	Mean (sec)	Standard Deviation	Confidence Interval (95%)
Native	49.275	0.983	49.275± 0.68
Docker	50.081	1.004	50.081± 0.71

Figure 5.15 shows that the CPU performance applications running in the Native and the Docker environments. The average execution time taken by the Docker environment is 50.081 seconds, which is very close to the 49.275 seconds of execution time by the Native environment. Table 5.7 shows the results of CPU performance where values of standard deviation and confidence interval of applications running in the Docker environment are very close to the results of applications running in the Native environment. In this case we can conclude that the container engine introduces a negligible impact on the CPU performance with reference to the Native environment. This is one of the advantages of using Docker as it is lightweight, and the overhead is negligible.

5.4.1.2 Memory I/O Performance

A Unix command, *mbw* has been used to test the Memory I/O performance. For testing Memory I/0 performance, *mbw* determines the available memory bandwidth by copying large arrays of data in memory, and performing three different tests (*memcpy*, *dumb*, and *mcblock*) [59]. The memory size is chosen to be 512 MB, since the total RAM of the Raspberry Pi 3 is 1 GB and the IoT Gateway applications and containers running over it consume approximately 500 MB. For *mcblock* test a random memory block size of 65,536 bytes is chosen by the system. Each test is conducted 20 times and the results are shown in Figure 5.16, and the standard deviation and confidence intervals (95%) are shown in Table 5.8

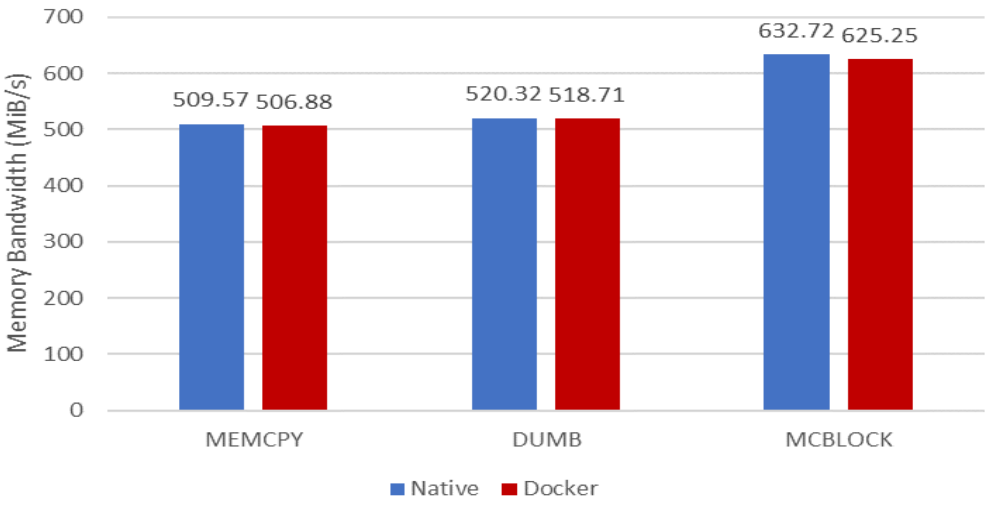

Figure 5.16 Memory Performance

Table 5.8 Results of Memory Performance

		Mean (MiB/s)	Standard Deviation	Confidence Interval (95%)
MEMCPY	**Native**	509.57	11.034	509.57 ± 14.52
	Docker	506.88	11.868	506.88 ± 14.67
DUMB	**Native**	520.32	10.137	520.32 ± 9.23

	Docker	518.71	10.904	518.71 ± 9.98
MCBLOCK	**Native**	632.72	14.298	632.72 ± 12.79
	Docker	625.25	13.869	625.25 ± 13.20

In all of the three tests, the memory bandwidth (MiB/s) of the Docker containers exhibits similar performance as that of the Native applications. The value of MCBLOCK test is noticeably higher than those of the other two because it copies memory by blocks. In Table 5.8, the Docker container approach displays slightly larger standard deviation and confidence interval, however the difference is very low that it can be neglected.

It can be deduced that the memory performance of the Docker containers is similar to that the native environment and thus introduces very negligible memory overhead.

5.4.1.3 Disk I/O Performance

The benchmark tool to measure disk I/O performance is *bonnie*++, which is aimed at characterizing the hard drive and filesystem by performing a number of simple tests [60]. This thesis conducts the tests of sequential output (by block), sequential input (by block) and random seeks on container and Native environments. The display of a higher speed of these tests indicates a better disk I/O performance of the system. To prevent the occurrence of caching, the recommended test file size should be at least double the RAM size of the system. Since the RAM size of Raspberry Pi 3 is 1GB, the test file size of the experiments in this thesis is chosen as 3GB. The performance results are shown in Figure 5.17 and Figure 5.18. Statistics of standard deviation and confidence intervals are shown in Table 5.9.

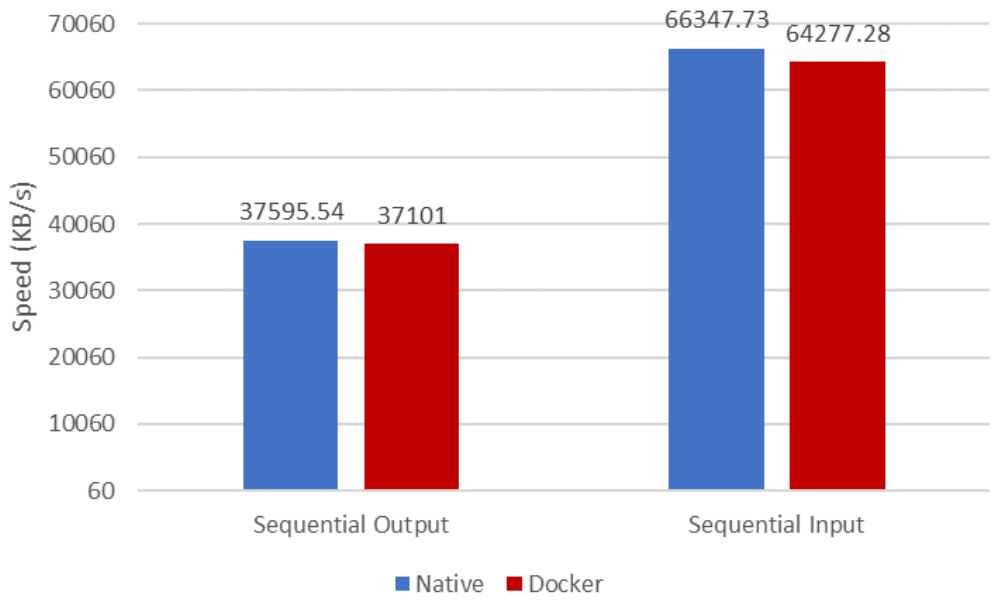

Figure 5.17 Sequential Input and Sequential Output Disk I/O Performance

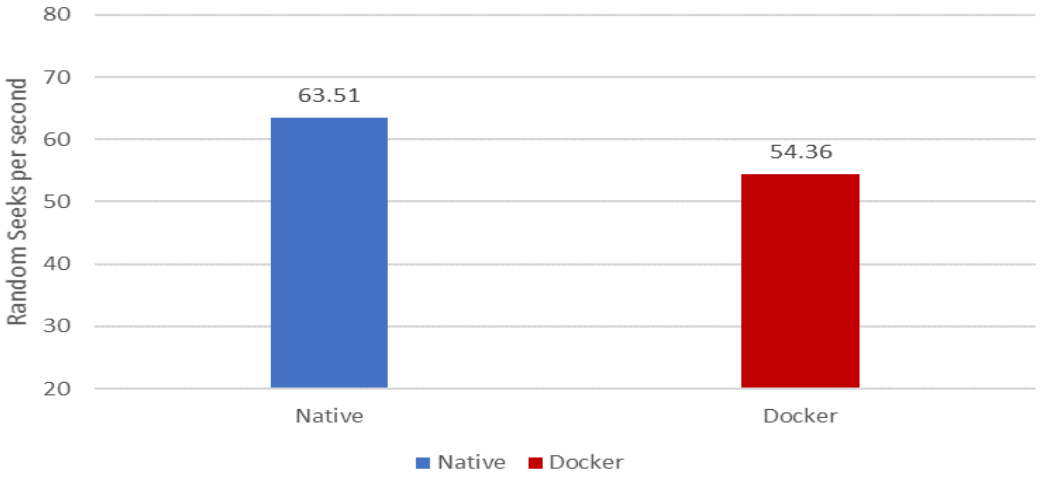

Figure 5.18 Random Seeks Disk I/O Performance

Table 5.9 Results of Disk I/O Performance (seconds)

		Mean	Standard Deviation	Confidence Interval (95%)
Sequential Output	**Native**	39595.81	1976.20	39595.81 ± 1414.62
	Docker	42014.27	1485.38	42014.27 ± 1062.58
Sequential Input	**Native**	61347.78	11663.51	61347.78 ± 30673.75
	Docker	75077.56	13364.37	75077.56 ± 37538.75

Random Seeks	**Native**	43.59	3.82	43.59 ± 2.87
	Docker	64.74	3.65	64.74 ± 2.15

From the above Figure 5.17 and Figure 5.18, we can observe that the differences are not substantial in the tests of sequential input and sequential output but have widened in test for random seeks. In the case of sequential input and output (see Table 5.9), all tests for the Native have a smaller standard deviation and confidence intervals than their Docker counterparts. However, in random seeks the standard deviation and confidence interval of the Native is larger than that of the Docker containers. Hence, even though the average speed of the Native is faster in random seeks but results for the Docker method show lower variations.

5.4.2 Application Benchmark Performance

The Application Benchmark aims to compare the performance of the IoT Gateway when the applications are running inside containers versus when applications are running on a native environment.

5.4.2.1 Memory Footprint Performance

Memory Footprint measures the total RAM utilized by the system applications and the operating system. *top command* [61] is a Linux command line utility to measure system performance. The experiment uses the *top* command to measure the memory usage of the whole system as well as memory consumed by each application in Native and the Docker containers and the results are shown in Figure 5.19.

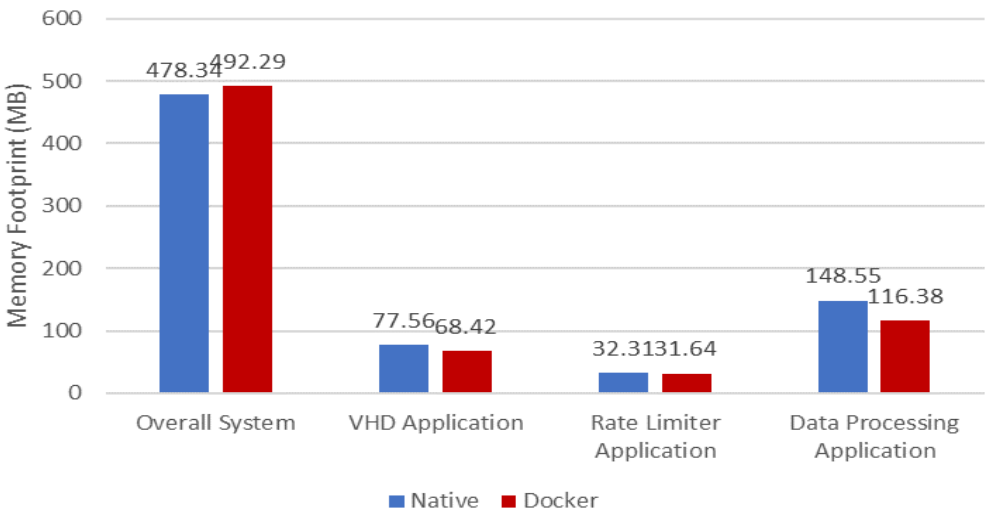

Figure 5.19 Memory Footprint (MB) of applications and overall system in native versus containerized environment

The whole system has a memory footprint of 478.34 MB in case of Native, while Docker containers occupy a slightly higher memory footprint of 492.29 MB due to memory consumed by the Docker engine. However, it is interesting to note that the memory footprints of each of Docker applications, i.e., VHD, Rate Limiter, or Data Processing application individually, has slightly low memory footprint than their native counterparts. The reason for this efficient memory usage by the Docker applications is due to the optimized configurations of the Docker environment, which does not run any unnecessary packages running in the background unlike applications running on the Native environment, hence Docker applications allow to save memory.

5.4.2.2 SQLite Database and Power Consumption Performance

This test is executed to benchmark the IoT Gateway database (SQLite) performance. A SQLite database application is created in the Native environment and another one is created as containerized application. Further, the number of concurrent threads is varied to evaluate

the application. Figure 5.20 depicts the results of the number of transactions per second with an increasing number of threads. The corresponding power consumption is also reported, which is measured using *powertop* , a Linux command line utility.

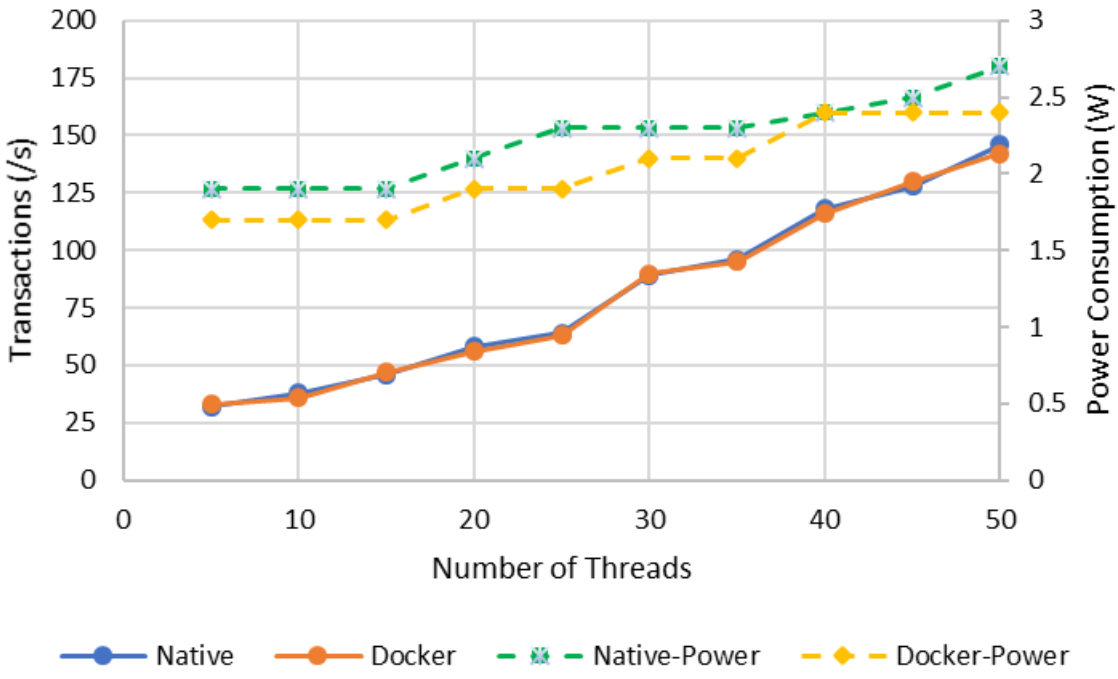

Figure 5.20 SQLite Transactions per second and power consumption versus. Concurrency in Native and containerized environment

From Figure 5.20, it can be observed that there is not a relevant difference in terms of database application performance between the Native and the Docker environments even after increasing the number of concurrent threads. However, the power consumption by both the Native and the Docker containerized database application shows a different trend. When the number of concurrent threads is increased, the maximum power consumption by the Native database application is 2.7 W which is 0.3 W higher than the power consumed by the Docker containerized database application. When a Docker container is executed,

the configuration information about the application running is specified together with any other dependency (e.g., libraries) and hence unnecessary system packages can be avoided which are installed by default on the Native environment. Removing dependency on any unnecessary package within a container helps in efficient usage of the power resource.

5.4.2.3 Boot-Up Time Performance

To measure the boot-up time, *systemd-analyze* [62], a Linux based command is used. The boot-up time of our proposed IoT solution is measured by running the MCVHD application on both the Native as well as the Docker containerized environments.

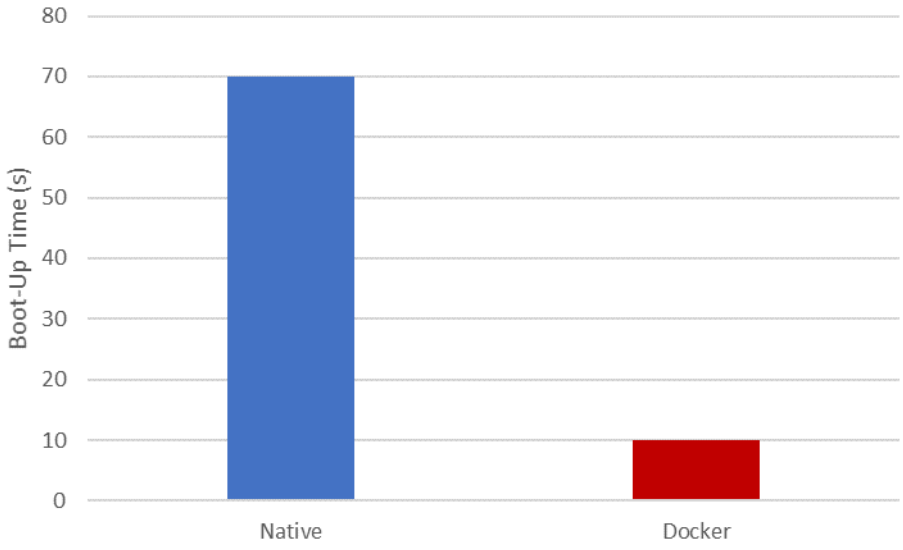

Figure 5.21 Boot - Up Time Performance

It is observed that the boot-up time of MCVHD application inside the Docker containerized environment is very low in comparison to that of Native counterpart. Figure 5.21, shows the significant difference in boot-up time that the Docker containerized MCVHD application takes, i.e., only 10 seconds in startup while the same MCVHD application inside the Native environment takes 70 seconds.

This shows the promising benefit of using the Docker containers as they are lightweight and have optimized file structure. We conclude from this experiment that the Docker containerized environment has significantly better performance in booting application due to its lightweight nature. Hence, the containerized application is well suited for resource constrained IoT application.

5.5 Chapter Summary

In this chapter, a performance evaluation of the proposed IoT solution was presented by conducting experiments in two phases. In the first phase, experiments were conducted to evaluate the proposed multi-criteria based VHD (MCVHD) algorithm in comparison with the conventional RSS Quality based VHD algorithm for vertical handover between the Wi-Fi, Radio and Satellite networks. Our thesis proposes a Fuzzy Logic based approach to convert linguistic network parameter values and weights to concrete numerical values. This approach makes our proposed MCVHD algorithm more efficient as shown by the experiment results. Our proposed algorithm outperforms the conventional RSS Quality based VHD algorithm by minimizing handover failures, unnecessary handovers and handover time.

In the second phase, experiments were conducted to evaluate the performance of our proposed IoT solution as a Docker containerized microservice architecture. System specific and application specific experiments were conducted which shows that Docker as a virtualization technology adds negligible overhead on the system. In a few experiments like application memory footprint, power consumption by application and boot-up time the Docker container application outperformed the Native application. Hence, based on the

experimental results, it could be concluded that Docker containers are lightweight and isolated and easy to deploy and could be efficiently used on resource constrained IoT devices like Raspberry Pi 3.

.

.

Chapter 6 Conclusion and Future Works

This chapter presents conclusion by reviewing the main contributions of the thesis in Section 6.1 and discusses future directions in Section 6.2.

6.1 Conclusion

Internet of Things (IoT) is a network paradigm in which computing, and networking capabilities are embedded in objects for interconnection. Many application areas of IoT are focused on mobility of objects, such as UAVs, drones, cars etc., and hence the need for mobility management in IoT. One of the challenges faced by mobility management in mobile IoT is vertical handover between heterogenous networks to provide seamless connectivity. This thesis proposed a containerized IoT solution for efficient vertical handover in mobile IoT.

The main scientific contribution of the thesis is the proposed multi-criteria vertical handover decision (MCVHD) algorithm for efficient IoT services in heterogeneous networks like Wi-Fi, Radio and Satellite network based. The proposed IoT solution takes into consideration the multiple criteria, such as RSS Quality, data service cost, data rate, latency, reliability, network coverage range, power consumption, and mobile terminal velocity to select best available network. This type of vertical handover decision (VHD) method based on analysis of multiple criteria, falls under the domain of Multiple-Attributed Decision Making(MADM) method. The thesis adopts the TOPSIS technique which is one of the MADM methods. Among the chosen multi-criteria, latency and reliability are two factors which cannot be determined precisely and are not possible to be handled by any of

the MADM methods. Hence, to deal with this imprecise data, the thesis enhances TOPSIS technique by proposing fuzzy logic based approach for multi-criteria VHD in IoT system for Wi-Fi, Radio and Satellite network.

To demonstrate the capability of the proposed IoT solution, an experimental testbed is prepared for the IoT system using JY901 sensor node, which measures acceleration, magnetic field and orientation and angular velocity, and stream data to Raspberry Pi 3 which acts as an IoT Gateway. The Raspberry Pi 3 is serially connected a Radio Modem and an Iridium SBD Satellite Modem for data transmissions to the base station.

The experiments were conducted to evaluate the performance of the proposed MCVHD algorithm in comparison with the conventional RSS Quality based VHD for vertical handover between Wi-Fi, Radio and Satellite networks. The experimental results showed that the proposed MCVHD algorithm outperformed the conventional RSS Quality based VHD in terms of reduced number of vertical handover, reduced vertical handover failures and reduced handover delay.

As the IoT devices are typically resource constrained, hence a major part of the thesis is also dedicated to analyzing a lightweight virtualization technology that is suitable for IoT devices. As a contribution, the thesis also presented the proposed IoT solution as a containerized microservices architecture for resource constrained IoT devices, e.g., Raspberry Pi 3 as IoT Gateway device. The proposed IoT solution employs Docker container for microservices. Each Docker container runs in loosely isolated environment, hence it produces negligible impact on other software components. Due to isolation, Docker provides benefits like easy application development, easy to ship and deploy and easy management of the applications without affecting other containers or applications.

Experiments were conducted to evaluate the performance of proposed containerized IoT solution on resource constrained IoT device like Raspberry Pi 3versus running same applications on a native environment. The experiments were conducted using synthetic benchmark and the results showed that Docker containers performed very closely to that on the native environment and produced negligible impact in terms of CPU, memory and disk I/O performance. Experiments were also conducted using application benchmarks for measuring memory footprint, boot-up time of each application and power consumption by SQLite database application running on IoT Gateway (Raspberry Pi 3). Interestingly, the memory footprint of individual application is found to be relatively less in the containerized environment than on the native environment. Container applications also outperformed in boot-up time and power utilization during SQLite database transactions. These findings demonstrate that Docker containers are lightweight, hence negligible impact on the system, and are suitable for resource constrained IoT devices like Raspberry Pi 3.

Sensors in IoT system generate and transmit enormous volume of data, which is a major challenge in terms of data transmissions over a network, storage capacity and data processing. Remote mobile IoTs often uses the satellite network when there is no other network coverage. Since satellite data transmission cost is very high and the data rate is low, hence such IoT system needs an efficient mechanism to handle data flow. To overcome this challenge in IoT system, this thesis also contributes by implementing a network function at IoT Gateway to limit the data rate according to the network chosen for transmission. The rate limiter network function works with MCVHD application and reduces the rate of data transmission whenever the network is switched from Wi-Fi or

Radio to Satellite by the mobile IoT. Hence, promising benefits of container technology along with the efficient MCVHD application and rate limiter network function make an efficient IoT solution for Mobile IoT.

6.2 Future Works

This thesis can be extended by enhancing the proposed IoT solution by adding more communication links like cellular network capability. Cellular network has a wider coverage and thus can minimize the handover to costlier network like Satellite.

The proposed MCVHD algorithm takes eight criteria into consideration. As a future work it can also be enhanced and tested by including more network and user parameters, e.g., packet loss, jitter, etc., are worth considering.

The proposed IoT solution implements rate limiter as a network function to limit rate as per the selected network by MCVHD. This feature efficiently minimizes the service cost by reducing data flow through satellite network. Though the proposed thesis implements network function for data rate, it does not address security of the IoT system and sensor nodes. Another potential future direction is to explore, other virtualized network function for security in IoT system can be investigated and implemented over the proposed IoT solution in this thesis.

The proposed IoT solution employs Wi-Fi, Radio and Satellite network, thus it potentially could provide connectivity across the globe. As the proposed IoT solution was tested with mobile IoT on the road but not tested with aerial vehicles such as UAVs or water vehicles like ships. Hence, extending the experiments to test the proposed IoT solution in other environments can give more useful insights.

References

[1] IEEE IoT Initiative , https://iot.ieee.org/images/files/pdf/IEEE_IoT_ Towards _Definition_ Internet_of_Things_Revision1_27MAY15.pdf. Last accessed on January 05, 2018.

[2] Exponential Growth Curve of IoT devices, https://dashbouquet.com/blog/web-development/the-most-promising-internet-of-things-trends-for-2018. Last accessed on January 01, 2018.

[3] A. S. Gaur, J. Budakoti, C. H. Lung and A. Redmond, "IoT-equipped UAV communications with seamless vertical handover," *Proceedings of the IEEE Conference on Dependable and Secure Computing*, 2017, pp. 459-465.

[4] Docker, https://www.docker.com. Last accessed on January 06, 2018.

[5] S. Vashi, J. Ram, J. Modi, S. Verma and C. Prakash, "Internet of Things (IoT): A vision, architectural elements, and security issues," *Proceedings of the International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC),* 2017, pp. 492-496.

[6] uCode, IETF Standard, https://tools.ietf.org/html/rfc6588, Last accessed on January 05, 2018.

[7] P. K. Kamma, C. R. Palla, U. R. Nelakuditi and R. S. Yarrabothu, "Design and implementation of 6LoWPAN border router," *Proceeding of the 13th International Conference on Wireless and Optical Communications Networks (WOCN)*, 2016, pp. 1-5.

[8] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, 2015, pp. 2347-2376.

[9] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained Application Protocol (CoAP).draft-ietf-core-coap-18," Internet Eng. Task Force (IETF), Fremont, CA, USA, 2013. Last accessed on January 07, 2018.

[10] D. Locke, "MQ telemetry transport (MQTT) v3. 1 protocol specification," IBM developerWorks, Markham, ON, Canada, Tech. Lib., 2010. [Online]. Available:Http://Www.Ibm.Com/Developerworks/Webservices/Library/Ws-Mqtt/Index.Html. Last accessed on January 05, 2018.

[11] "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0," Adv. Open Std. Inf. Soc. (OASIS), Burlington, MA, USA, 2012. Last accessed on January 02, 2018.

[12] NFV-IoThttps://www.ietf.org/proceedings/91/slides/slides-91-sdnrg-3.pdf., Last accessed on 05, January.

[13] Global IoT in Transport Market, http://www.satprnews.com/2017/08/29/global-iot-in-transportation-market-2022-research-report/. Last accessed on 05, January.

[14] S. Mohanty and I. F. Akyildiz, "A cross-layer (layer 2 + 3) handoff management protocol for next-generation wireless systems," *IEEE Transactions on Mobile Computing*, vol. 5, no. 10, 2006, pp.1347–1360.

[15] X. Yan, Y. A. Şekercioğlu, and S. Narayanan, "A survey of vertical handover decision algorithms in Fourth Generation heterogeneous wireless networks," *Computer Networks*, vol. 54, no. 11, 2010, pp. 1848–1863.

[16] C. Chi, X. Cai, R. Hao, F. Liu, “Modeling and analysis of handover algorithms,” *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, 2007, pp. 4473–4477

[17] A. H. Zahran, B. Liang, and A. Saleh, “Signal threshold adaptation for vertical handoff in heterogeneous wireless networks,” *Mobile Networks and Applications*, vol. 11, no. 4, 2006, pp. 625–640.

[18] R. Tawil, G. Pujolle, and O. Salazar, “A vertical handoff decision scheme in heterogeneous wireless systems,” *Proceedings of the 67th Vehicular Technology* Conference (VTC), 2008, pp. 2626–2630.

[19] J. Hou, D.C. O’Brien, “Vertical handover-decision-making algorithm using fuzzy logic for the integrated radio-and-OW system,” *IEEE Transactions on Wireless Communications*, vol. 5, no. 1, 2006, pp. 176–185.

[20] H. Liao, L. Tie, Z. Du, “A vertical handover decision algorithm based on fuzzy control theory*,” Proceedings of the 1st International Multi Symposiums on Computer and Computational Sciences (IMSCCS)*, 2006, pp. 309–313.

[21] S. Luo, Z. Lin, X. Chen, Z. Yang and J. Chen, "Virtualization security for cloud computing service," *Proceeding of the International Conference on Cloud and Service Computing,* 2011, pp. 174-179.

[22] Type 1 and Type 2 Hypervisors, http://vapour-apps.com/what-is-hypervisor, Last accessed on December 01, 2017.

[23] Container Based Virtualization, https://www.redhat.com/en/topics/containers/whats-a-linux-container, Last accessed on December 21, 2017.

[24] Y. C. Tay, K. Gaurav and P. Karkun, "A Performance Comparison of Containers and Virtual Machines in Workload Migration Context," *Proceeding of the IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2017, pp. 61-66.

[25] S. Nadgowda, S. Suneja and A. Kanso, "Comparing Scaling Methods for Linux Containers," *Proceeding of the IEEE International Conference on Cloud Engineering (IC2E)*, 2017, pp. 266-272.

[26] A.H. Zahran, B. Liang, "Performance evaluation framework for vertical handoff algorithms in heterogeneous networks," *Proceedings of the IEEE International Conference on Communications (ICC)*, 2005, pp. 173–178.

[27] E. Stevens-Navarro, V.W.S. Wong, "Comparison between vertical handoff decision algorithms for heterogeneous wireless networks," *Proceedings of the 63rd Vehicular Technology Conference (VTC),* 2006, pp. 947–951.

[28] A. A. Yussuf, W. H. Hassan and S. Issa, "A review of VHD approaches in next generation wireless networks," *Proceedings of the 2nd International Conference on Digital Information and Communication Technology and it's Applications (DICTAP),* 2012, pp. 363-367.

[29] Gongye Ren, Jihong Zhao and Hua Qu, "A User Mobility Pattern based Vertical Handoff Decision algorithm in Cellular-WLAN Integrated Networks," Proceedings *of the 2nd IEEE International Conference on Computer and Communications (ICCC),* 2016, pp. 1550-1554.

[30] R. Tamijetchelvy and G. Sivaradje, "An optimal vertical handover for heterogeneous networks based on IEEE 802.21 MIH standards," *Proceedings of the 5th International Conference on Advanced Computing (ICoAC),* 2013, pp. 446-451

[31] Ismail A, Roh B-H, "Adaptive handovers in heterogeneous networks using fuzzy MADM," *Proceeding of the international conference on mobile IT convergence (ICMIC)*, 2011, pp. 99-104.

[32] A. Agrawal, A. Jeyakumar and N. Pareek, "Comparison between vertical handoff algorithms for heterogeneous wireless networks," *Proceedings of the International Conference on Communication and Signal Processing (ICCSP),* 2016, pp. 1370-1373.

[33] U. Baroudi and F. Al-Nasser, "Performance evaluation study on a multiple-parameter handoff algorithm," *Proceedings of the International Conference on Information Networking (ICOIN),* 2011, pp. 273-277

[34] A. A. Bathich, M. D. Baba and R. Ab Rahman, "SINR based Media Independent Handover in WiMAX and WLAN networks," *Proceedings of the IEEE International Conference on Computer Applications and Industrial Electronics (ICCAIE),* 2011, pp. 331-334.

[35] Qing He, "A novel vertical handoff decision algorithm in heterogeneous wireless networks," *Proceedings of the IEEE International Conference on Wireless Communications, Networking and Information Security*, 2010, pp. 566-570.

[36] H. M. Tahir, A. H. Al-Ghushami and Z. Ridzuan Yahya, "Selection of access network using cost function method in heterogeneous wireless network," *Proceedings of the*

International Conference on Multimedia Computing and Systems (ICMCS), 2014, pp. 789-793.

[37] M. Mansouri and C. Leghris, "A comparison between fuzzy TOPSIS and fuzzy GRA for the vertical handover decision making," *Proceedings of the Intelligent Systems and Computer Vision (ISCV),* 2017, pp. 1-6.

[38] Ganz F, Barnaghi P, Carrez, "Information Abstraction for Heterogeneous Real World Internet Data," *IEEE Sensors Journal*, vol. 13, no. 10, 2013, pp. 3793-3805.

[39] M. Lahby, C. Leghris, A. Adib, "An Enhanced-TOPSIS Based Network Selection Technique for Next Generation Wireless Networks," *Proceedings of the 20th International Conference on Telecommunications (ICT),* 2013, pp. 1-5.

[40] O. Semenova and A. Semenov, "The neuro-fuzzy controller for handover operation in mobile networks," *Proceedings of the IEEE 1st Ukraine Conference on Electrical and Computer Engineering (UKRCON),* 2017, pp. 806-812.

[41] Hung, C.C.; Chen, L.H, "A fuzzy TOPSIS decision making model with entropy weight under intuitionistic fuzzy environment," *Proceedings of the International MultiConference of Engineers and Computer Scientists,* 2009, pp. 1–4.

[42] M. Abid, T. A. Yahiya and G. Pujolle, "A Utility-based Handover Decision Scheme for Heterogeneous Wireless Networks," *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC),* 2012, pp. 650-654.

[43] Xiaoping Qiu, Ming Jian, Fei Wei and Yingbo Li, "A new decision-making method based on a typical utility function," *Proceedings of the International Conference on Machine Learning and Cybernetics*, 2013, pp. 1807-1811.

[44] Preeth E N, F. J. P. Mulerickal, B. Paul and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," *Proceeding of the International Conference on Control Communication & Computing India (ICCC),* 2015, pp. 697-700.

[45] Jy901, https://www.scribd.com/document/276842509/JY901-gyroscope-User-Manual-by-Elecmaster. Last accessed on January 09, 2018.

[46] Raspberry Pi 3, https://www.raspberrypi.org/products/raspberry-pi-3-model-b. Last accessed on January 09, 2018

[47] SQLite Database, https://www.sqlite.org.Last accessed on January 09, 2018

[48] Wi-Fi IEEE 802.11 standards, http://www.ieee802.org/11. Last accessed on January 09, 2018

[49] XTend Radio Modem, https://www.digi.com/pdf/ds_xtend.pdf. Last accessed on January 09, 2018

[50] USB to RS232 Converter, http://www.ftdichip.com/Products/Cables/USBRS232.htm Last accessed on January 09, 2018

[51] RockBLOCKMk2 Satellite Modem, http://www.rock7mobile.com/downloads /RockBLOCK -Product-Information-Sheet.pdf. Last accessed on January 09, 2018

[52] Dweet.io, https://dweet.io. Last accessed on January 09, 2018.

[53] A. Sill, "The Design and Architecture of Microservices," *IEEE Cloud Computing*, vol. 3, no. 5, 2016, pp. 76-80.

[54] A. Celesti, D. Mulfari, M. Fazio, M. Villari and A. Puliafito, "Exploring Container Virtualization in IoT Clouds," *Proceedings of the IEEE International Conference on Smart Computing (SMARTCOMP),* 2016, pp. 1-6.

[55] Freeboard.io, https://freeboard.io. Last accessed on January 02, 2018.

[56] S. Folea, D. Bordencea, C. Marcu and H. Valean, "Indoor localization based on Wi-Fi parameters influence," *Proceedings of the 36th International Conference on Telecommunications and Signal Processing (TSP),* 2013, pp. 190-194.

[57] Powertop, https://docs.oracle.com/cd/E36784_01/html/E36871/powertop-1m.html. Last accessed on January 02, 2018.

[58] Sysbench, https://github.com/akopytov/sysbench. Last accessed on January 02, 2018.

[59] Mbw, https://github.com/raas/mbw. Last accessed on January 01, 2018.

[60] Bonnie++, http://www.coker.com.au/bonnie++/. Last accessed January 02, 2018.

[61] Top, http://man7.org/linux/man-pages/man1/top. Last accessed January 03, 2018.

[62] Systemd-analyze,http://manpages.ubuntu.com/manpages/zesty/man1/systemd-analyze.1.html. Last accessed December 08, 2017.

[63] RSS-Quality, https://www.speedguide.net. Last accessed on 05 January 2017

[64] M. Lahby and A. Sekkaki, "Optimal vertical handover based on TOPSIS algorithm and utility function in heterogeneous wireless networks," *Proceedings of the International Symposium on Networks, Computers and Communications (ISNCC)*, 2017, pp. 1-6.

[65] Edge Gateway and Field Gateway, https://thenewstack.io/tutorial-prototyping-a-sensor-node-and-iot-gateway-with-arduino-and-raspberry-pi-part-1. Last accessed on 05 January 2017

[66] S. Chakraborty and C. H. Yeh, "A simulation comparison of normalization procedures for TOPSIS," *Proceedings of the International Conference on Computers & Industrial Engineering*, 2009, pp. 1815-1820.

Appendices

Appendix A

Docker files related to data communications using Wi-Fi, Radio, Satellite, Gateway node data processing, database process, visualization are presented below.

```
Dockerfile for IoT GateWay Server Node

# Pull Rasbian Debian OS base image for Raspberrry Pi

FROM resin/rpi-raspbian:jessie

MAINTAINER Amit Singh Gaur <amitgaur@cmail.carleton.ca>

# Install dependencies

RUN apt-get update && apt-get install -y \
   git-core \
   build-essential \
   gcc \
   python \
   python-dev \
   python-pip \
   python-virtualenv \
   python-pip cron\
   python-dev
   --no-install-recommends && \
   rm -rf /var/lib/apt/lists/*
RUN pip install pyserial
#Installing LEMP server

RUN apt-get -y install \

        nginx \

        php5 \

        php5-fpm
```

```
#Installing SQLite DB

RUN pip install sqlite3

RUN rm -rf /var/www/*

RUN mv /etc/nginx/sites-available/default /etc/nginx/sites-available/default.bak

COPY nginx-config /etc/nginx/sites-available/default

#Installing python requirements

RUN pip install pyserial simplejson configobj psutil gitpython

#Setting up users and permissions

RUN useradd -m -k /dev/null -G www-data

RUN mkdir -p /var/www/html

RUN chown -R www-data:www-data /var/www/html

RUN chown -R nginx:www-data /var/lib/nginx

RUN chown -R brewpi:brewpi /home/pi/dashboard

RUN find /home/brewpi -type f -exec chmod g+rwx {} \;

RUN find /home/brewpi -type d -exec chmod g+rwxs {} \;

RUN find /var/www -type d -exec chmod g+rwxs {} \;

RUN find /var/www -type f -exec chmod g+rwx {} \;

#Step 6 Cron Job for complete automation

COPY brewpi-crontab /etc/cron.d/brewpi

RUN chmod 775 /etc/cron.d/brewpi

RUN chown -R nginx:www-data /var/lib/nginx

EXPOSE 80 443

# Start the cron daemon shell
```

```
COPY configure-and-start.sh configure-and-start.sh

RUN chmod +x configure-and-start.sh

CMD ./configure-and-start.sh

# Define working directory

WORKDIR /etc/nginx

mkdir -p /home/pi/data

WORKDIR /data

VOLUME /data

# Copy the current directory contents into the container at /DataAggregator

ADD   .  / DataAggregator
# Add the Python DataAggregator.py to the Dockerfile:

ADD  DataAggregator.py /

# Run DataAggregator.py and Dashboard. php when the container launches

CMD ["bash","nginx", "Dashboard.php", "--port", "443"]

CMD [ "python", "./DataAggregator.py" ]
```

Dockerfile for Failover Algorithm running at Gateway Node

```
# Pull Rasbian Debian OS base image for Raspberrry Pi

FROM resin/rpi-raspbian:jessie

MAINTAINER Amit Singh Gaur <amitgaur@cmail.carleton.ca>

# Install dependencies

RUN apt-get update && apt-get install -y \
    git-core \
    build-essential \
    gcc \
```

```
    python \
    python-dev \
    python-pip \
    python-virtualenv \
    python-pip cron\
   python-dev
    --no-install-recommends && \
    rm -rf /var/lib/apt/lists/*
RUN pip install pyserial

#Step  Cron Job for complete automation

COPY /home/pi/Failover -crontab /etc/cron.d/Failover

RUN chmod 775 /etc/cron.d/Failover

RUN chown -R nginx:www-data /var/lib/nginx

EXPOSE 80 443

# Start the cron daemon shell

COPY configure-and-start.sh configure-and-start.sh

RUN chmod +x configure-and-start.sh

CMD ./configure-and-start.sh

mkdir -p /home/pi/Failover

# We copy just the requirements.txt first to leverage Docker cache

COPY ./requirements.txt /Failover/requirements.txt
WORKDIR /Failover
RUN pip install -r requirements.txt
VOLUME /Failover
# Copy the current directory contents into the container at /Failover
ADD   .  / Failover
COPY . / Failover
ENTRYPOINT [ "python" ]
# Add the Python Failover.py to the Dockerfile:

ADD  Failover.py /

# Run Failover.py when the container launches
```

```
CMD [ "python", "./Failover.py", "--port", "443"]
```

Dockerfile for Wifi Algorithm running at Gateway Node

```
# Pull OS base image for Raspberrry Pi and other dependencies from Failover image

FROM Failover

EXPOSE 80 445

# Start the cron daemon shell

COPY configure-and-start.sh configure-and-start.sh

RUN chmod +x configure-and-start.sh

CMD ./configure-and-start.sh

RUN mkdir -p /home/pi/Failover/Wi-Fi

RUN cd /home/pi/Failover/Wi-Fi

# We copy just the requirements.txt first to leverage Docker cache

COPY ./requirements.txt /Failover/ Wi-Fi requirements.txt
WORKDIR / Wi-Fi
RUN pip install -r requirements.txt
VOLUME / Wi-Fi
# Copy the current directory contents into the container at /Wi-Fi
ADD   .  / Wi-Fi
COPY . / Wi-Fi
ENTRYPOINT [ "python" ]
# Add the Python Wi-Fi.py to the Dockerfile:

ADD  Wi-Fi.py /

# Run Wi-Fi.py when the container launches

CMD [ "python", "./Failover.py", "--port", "445"]
```

Dockerfile for Python Algorithm for sending Data through Radio Modem running at Gateway Node

```
# Pull OS base image for Raspberrry Pi and other dependencies from Failover image

FROM Failover

EXPOSE 80 446

# Start the cron daemon shell

COPY configure-and-start.sh configure-and-start.sh

RUN chmod +x configure-and-start.sh

CMD ./configure-and-start.sh

RUN mkdir -p /home/pi/Failover/Radio

RUN cd /home/pi/Failover/Radio

# We copy just the requirements.txt first to leverage Docker cache

COPY ./requirements.txt /Failover/ Radio requirements.txt
WORKDIR / Radio
RUN pip install -r requirements.txt
VOLUME / Radio
# Copy the current directory contents into the container at /Radio
ADD   .  / Radio
COPY . / Radio
ENTRYPOINT [ "python" ]
CMD [ "Radio.py" ]
# Add the Python Radio.py to the Dockerfile:

ADD  Radio.py /

# Run  Radio.py when the container launches

CMD [ "python", "./Radio.py", "--port", "446"]
```

Dockerfile for Python Algorithm for sending Data through Satellite Modem running at Gateway Node

```
# Pull OS base image for Raspberrry Pi and other dependencies from Failover image

FROM Failover

EXPOSE 80 447

# Start the cron daemon shell

COPY configure-and-start.sh configure-and-start.sh

RUN chmod +x configure-and-start.sh

CMD ./configure-and-start.sh

RUN mkdir -p /home/pi/Failover/Satellite

RUN cd /home/pi/Failover/Satellite

# We copy just the requirements.txt first to leverage Docker cache

COPY ./requirements.txt /Failover/ Satellite requirements.txt
WORKDIR / Satellite
RUN pip install -r requirements.txt
VOLUME / Satellite
# Copy the current directory contents into the container at /Satellite
ADD   .  / Satellite
COPY . / Satellite
ENTRYPOINT [ "python" ]
# Add the Python Radio.py to the Dockerfile:

ADD  Satellite.py /

# Run  Radio.py when the container launches

CMD [ "python", "./Satellite.py", "--port", "447"]
```

Appendix B

Appendix B presents Compose YAML code file. Compose YAML file defines the simplified and consolidated version of orchestrated multi-container microservices for proposed Containerized Vertical Handover Solution for IoT presented in Appendix B.

```
version: '3'
services:
  Multi Criteria Vertical-Handover:
    build: ./Failover
      context: .
      dockerfile: Dockerfile
    image: Vertical_Handover/ Failover
    deploy:
      resources:
        limits:
          cpus: '0.50'
          memory: 50M
        reservations:
          cpus: '0.25'
          memory: 20M
    ports:
      - "9200:9200"
    container_name: Vertical_Failover
    restart: on-failure
    restart: always
    environment:
    - DEBUG=1
    links:
      - Multi Criteria Vertical-Handover: Multi Criteria Vertical-Handover
    volumes:
      - .:/code
      - /home/pi/VerticalHandover
    privileged: true
    Wi-Fi Mode:
    build: ./Wi-Fi
      context: .
      dockerfile: Dockerfile

    image: Wi-Fi Mode Container
    deploy:
      resources:
```

```
    limits:
      cpus: '0.10'
      memory: 20M
    reservations:
      cpus: '0.15'
      memory: 10M
ports:
  - "9200:9200"
depends_on:
  - Multi Criteria Vertical-Handover
container_name: Wi-Fi_Mode_Container
restart: on-failure
restart: always
environment:
- DEBUG=1
links:
  - Wi-Fi Mode: Wi-Fi_Mode_Container
volumes:
  - .:/code
  - /home/pi/Handover/Wi-Fi
privileged: true
Radio Mode:
build: ./Radio
   context: .
   dockerfile: Dockerfile
image: Radio_Mode_Container
deploy:
  resources:
    limits:
      cpus: '0.10'
      memory: 20M
    reservations:
      cpus: '0.15'
      memory: 10M
ports:
  - "9200:9200"
depends_on:
  - Wi-Fi Mode
container_name: Radio_Mode_Container
restart: on-failure
restart: always
environment:
- DEBUG=1
links:
  - Radio Mode: Radio_Mode_Container
volumes:
```

```
  - .:/code
  - /home/pi/Failover/Radio
 privileged: true
 Satellite Mode:
 build: ./Satellite
    context: .
    dockerfile: Dockerfile
 image: Vertical_Handover/ Radio_Mode_Container
 deploy:
  resources:
    limits:
      cpus: '0.10'
      memory: 20M
    reservations:
      cpus: '0.15'
      memory: 10M
 ports:
  - "9200:9200"
 depends_on:
  - Multi Criteria Vertical-Handover
 container_name: Satellite_Mode_Container
 restart: on-failure
 restart: always
 environment:
 - DEBUG=1
 links:
  - Radio Mode: Satellite_Mode_Container
 volumes:
  - .:/code
  - /home/pi/Failover/Satellite
 privileged: true
 IoT-DB:
  image: sqlite3:latest
  volumes:
    - db_data:/var/lib/sqlite
  restart: always
  environment:
    DB_ROOT_PASSWORD: sensordb
    DB_DATABASE: sensordb

IoT Gateway Processing Node:
 build: ./Data-Aggregator
    context: .
    dockerfile: Dockerfile

 image: Data-Aggregator / IoT-Gateway
```

```
 deploy:
   resources:
     limits:
       cpus: '0.50'
       memory: 200M
     reservations:
       cpus: '0.50'
       memory: 100M
 ports:
   - "9200:9200"
 depends_on:
   - IoT-DB
 container_name: Data-Aggregator
 restart: on-failure
restart: always
environment:
 - DEBUG=1
links:
   - Data-Aggregator: Data-Aggregator / IoT-Gateway
volumes:
   - .:/code
   - /home/pi/Data-Aggregator
privileged: true

IoT-Dashboard:
 build: ./IoT-Dashboard
    context: .
    dockerfile: Dockerfile
 image: IoT-Dashboard
 deploy:
   resources:
     limits:
       cpus: '0.10'
       memory: 20M
     reservations:
       cpus: '0.10'
       memory: 10M
 container_name: IoT-Dashboard
 restart: always
 depends_on:
    - IoT-DB
    - IoT Gateway Processing Node
  ports:
    - "8000:80"
  restart: always
  environment:
```

```
  - DEBUG=1
  - ConnectionString=IoT-DB.data
links:
  - IoT-Dashboard : IoT-Gateway
volumes:
  - .:/code
  - /home/pi/IoT-Dashboard
privileged: true
```