Reliable Edge

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering
by Research

by

Jitender Grover
20162313
jitender.grover@research.iiit.ac.in

International Institute of Information Technology
(Deemed to be University)
Hyderabad - 500032, INDIA
June 2019

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled ”Reliable Edge” by Jitender Grover, has been carried out under my supervision and is not submitted elsewhere for a degree.

29-June-2019
Date

Adviser: Dr. Rama Murthy Garimella

To My Parents

Acknowledgments

First and foremost, I would like to thank my thesis advisor Dr. Rama Murthy Garimella. His guidance, knowledge and support has helped me throughout my MS journey. I am grateful to him for allowing me to work in a futuristic research area and helping me in contributing to the present set of knowledge in this area. I count my decision to join IIIT-H under his guidance as one of the best decisions of my lifetime. He has played a key role in moulding my personality and steering my thoughts in the correct direction.

Next, I would like to acknowledge the contribution of Mr Rhishi with his valuable discussions and suggestions on the part of research, we completed together. Special thanks to Mr Vikas Chouhan (IIT Roorkee), Mr Suman (IIIT Hyderabad) and Ms Shivangi Katiyar for the fruitful discussions and valuable suggestions throughout this journey. I thank Prof. Atul Sharma (UIET, Kurukshetra University, Kurukshetra) for his timely doses of motivation and encouragement.

Other than research contributors, I am thankful to my buddies Suman, Prateek, Durga, RK Syam, Anish, Pulkit Verma, Pulkit Velani, Ganesh, Yaswanth, Mahesh, Zakir, Sukesh, Krishna, Jayanth, Huzaifa and Bhaskar for being an integral part of my life at IIIT-H. Thanks a lot for sharing joyful moments throughout this tenure. While involved in extra-curricular activities and volunteering, I got great support from Prof. Syed, Prof. Rawat and Mr. Ramana (AR) for IEEE Students' Branch; Prof. Suvarna and Prof. Priyanka for Students' Mental Health related issues; Prof. Avinash, Mr Srikanth, Mr Pulkit Velani, Mr Pruthwik, Mr Suman and Mr Prateek for Yuktahar mess related issues; Prof. Radhika, Mr Prabhakar M. and Mr Yaswanth Naidu for things related to campus and hostel life. I am thankful to all of them for making my volunteering journey joyful and productive at IIIT. They have been a relentless source of encouragement and motivation throughout. Thanks to Sailaja ma'am for her help and support in administrative works.

I also thank Dr Debi Prasad Kanungo, Professor, CSIR-CBRI, Roorkee, India; Dr Rajib Panigrahi, Associate Professor, ECE, IIT Roorkee, India; and Dr Sateesh K. Peddoju, Associate Professor, CSE, IIT Roorkee, India for allowing me to work on the NMHS funded Landslide Early Waring System project (Project ID: NMHS/2017-18/MG47/31) and for allowing me to utilize the SAI Lab facilities for the implementation of the proposed work.

Last but not the least, I express my very profound gratitude to my parents and wife Ms Anjali Grover for believing in me and supporting me to fulfill my dreams.

Abstract

IoT domains such as health-care, automation and control, augmented and virtual reality etc are incubating novel applications and devices everyday. The data generated in these domains must be processed under strict delay requirements. Beyond the processing deadlines, the information loses its value. Apart from that, processing resources are required to be available all the time and Cloud's or communication failure must not hinder the services provided by such IoT applications. So, *reliability* is one of the major concerns for all these IoT applications. Considering all these requirements, it is been observed that instead of processing such data at the far Cloud, it is beneficial to process it closer to the source. This led to a new paradigm called Edge Computing. Edge computing has emerged as an effective solution for delay sensitive IoT applications. In the Edge-Cloud hierarchy, reliability and fault tolerance are the major issues. This thesis proposes a novel fault-tolerant and reliable hierarchical IoT-Cloud architecture which can survive the failures of the Cloud Server and the Edge Server(s).

In the proposed architecture, the sensed data processing is distributed over four levels (Cloud-Fog-Mist-Dew) based on the level processing power and distance from the end IoT devices. This hierarchical architecture is an advancement over the existing IoT-Cloud architecture. The proposed system becomes reliable by replicating the model-files (generated after data training) and some relevant data at the Edge Server from the Cloud. It allows the Edge to generate feedback after receiving the sensed data, in case of any event happens. It indeed improves the reliability of IoT applications by providing them services in the case of unavailability of the processing resources especially due to the communication failure with the Cloud Server.

Although, hierarchical Edge-Cloud architecture resolves the problems of resource unavailability and feedback delays, what if the Edge Server fails? It induces unheard issues of reliability and fault tolerance at the Edge. The system can not be completely reliable until the reliability issues at the edge of the network are resolved. This thesis proposes a novel mechanism using connection switching to deal with Edge Servers' failures. In case of an Edge failure, IoT application is redirected to an alternate available Edge Server. If there are multiple alternate Edge Servers available, redirection to a new Edge is decided based on the delay-tolerance of the IoT applications. It makes the entire system reliable and fault-tolerant.

The proposed IoT-Cloud architecture with Reliable-Edge has been implemented as an extension of Landslide Early Warning System (LEWS) to improve its reliability. LEWS is one of the most suitable applications to apply the proposed mechanism(s). During laboratory experiments, results demonstrate

that the system attains maximum availability of the computation resources. Along with this, results prove its efficiency when the proposed system is analyzed theoretically and simulated using Matlab.

The thesis also contributes to the construction of a Reliable Edge Controller (EC) that reliably assigns any type of computational resources available at the edge to the IoT applications. A variety of devices available at the network access layer have been considered to be utilized to serve IoT applications. This includes the use of devices available with private users, dedicated Edge Servers and Cloud infrastructure. The proposed system learns the optimal operating parameters during initial runs. Using the knowledge acquired in the learning phase, an integer linear programming problem is formulated to minimize the Mean Time To Complete (MTTC) the request for all the IoT nodes. The solution of the formulated problem provides optimized resource allocation for all the IoT nodes. Later, considering the unreliable nature of the privately owned devices, the learning and formulation has been extended to incorporate probability of failure of these devices. This evolves the EC into a reliable one.

Keywords: Internet of Things (IoT), Edge Computing, Real-time Applications, Reliability, Fault Tolerance, Cloud Computing, SDN

Contents

List of Abbreviations

IoT	Internet of Things
LEWS	Landslide Early Warning System
ILP	Integer Linear Programming
EC	Edge Controller
ISP	Internet Service Provider
DCA	Dynamic Channel Allocation
MA	Mobile Agent
CAGR	Compound Annual Growth Rate
ICU	Intensive Care Unit
CH	Cluster Head
P2P	Peer-to-Peer
AR	Augmented Reality
VR	Virtual Reality
VANET	Vehicular Ad-Hoc Network
SDN	Software Defined Network
OF	OpenFlow
ReST	Representational State Transfer
MQTT	Message Queuing Telemetry Transport
CoAP	Constrained Application Protocol
5G	Fifth Generation
API	Application Program Interface
WAN	Wide Area Network
FIFO	First Come First Out
IP	Internet Protocol
QoS	Quality of Service
CN	Core Network
AZ	Availability Zone
GA	Genetic Algorithm
CoRE	Constrained ReSTful Environment
RDP	Resource Discovery Protocol

RR	Round Robin
MTTC	Mean Time To Complete
HTTP	HyperText Transfer Protocol
WLAN	Wireless Local Area Network
4G	Forth Generation
ML	Machine Learning
GSM	Global System for Mobile
LTE	Long Term Evolution
OS	Operating System
GB	Gigabytes
GPU	Graphics Processing Unit
GDDR	Graphics Double Data Rate
RAM	Random Access Memory
USB	Universal Serial Bus
RPi	Raspberry Pi
Wi-fi	Wireless Fidelity

List of Symbols

Chapter 2

$TypeA$	Type A Edge Computing Device
$TypeA$	Type B Edge Computing Device
C_A	Cost of Type A Edge Computing Device
C_B	Cost of Type B Edge Computing Device
a_ij	Existence of Type A Edge Computing Device at coordinates i, j
b_ij	Existence of Type B Edge Computing Device at coordinates i, j
M	Number of rows in rectangular grid
N	Number of columns in rectangular grid
D_γ	mutually disjoint sets to hold MxN locations on grid
S	Number of high-end edge servers are required as a constraint for each set D_γ
S_1	Number of Type A high-end edge servers are required as a constraint
S_2	Number of Type B high-end edge servers are required as a constraint
P_i	Probability Mass Function
$n_i^{\theta}s$	Historical traffic in the spectrum bands in cells $fn_1, n_2, n_3, ..., n_Mg$
L	Total number of available channels
Z	Random variable associated with number of channels allocated per cell
$E[Z]$	Expectation of Z that is required to be maximized
$q_1 \quad q_2 \quad ... \quad q_M$	Order of Probabilities

Chapter 3

A_t	Availability
R	Reliability
F_p	Failure Probability
N	Number of total Edge Nodes
M	Number of unavailable Edge Nodes
T	Maximum Number of Trials

Chapter 4

m	Number of serving nodes

n	Number of IoT nodes
a_i	Maximum serving capacity of serving node i
b_j	Maximum expected requests from IoT node j
t_{ij}	Time Taken for processing a request between node i and request j

List of Figures

List of Tables

Chapter 1

Introduction

Internet of Things (IoT) has the internet as a network of physical devices used in everyday life. There is a wide range of IoT devices available such as blinds on your window, door locks, taps, machines in the assembly line factory, traffic signals in Smart City etc. When physical things are augmented with computing, connectivity, sensors and actuators, it can be called IoT. IoT allows us to connect and monitor these physical devices 24x7, analyse their usage and conditions to control them in a better way. It helps in creating more valuable output for the users.

Physical things can sense, communicate and collaborate together to create a greater intelligent outcome to bring more value to the users of those things. The first class of beneficiaries is *users*. Users can monitor and control devices remotely and efficiently. It increases the service life of the devices and users can also view devices to be operated in an autonomous way. The second class of beneficiaries *manufacturers* or *administrators*. They can get real-time insights to their devices' location, condition, usage and performance. It helps them to improve the quality of their future products. They can also take decisions on upgrading the existing products in use to improve their lifetime and performance.

Broadly IoT can be classified into three categories namely *Consumer IoT, Industrial IoT* and *Civic IoT*. *Consumer IoT* includes personal and pet monitoring, smart homes where everyday appliances such as door locks, kitchen etc can be connected, smart buildings where electric metre, lights, elevators, water supply etc can also be connected together. Whereas, in *Industrial IoT* different machines in a particular factory can be connected together. One example can be a wind farm where wind turbines connected to the internet being monitored remotely. Another example can be medical devices in the hospital which are connected to manage patients in a better way. The third category is *Civic IoT* which includes which includes smart public services such as rail, road, air transportation, electric supply grid, water supply system or any other public services.

Cloud Computing [1] has been used for IoT data processing, storage, analytics and control actions for the actuators. Cloud receives the sensed data through gateways via the core internet. A user can control any of the connected IoT devices through the Cloud interface from any remote location. Security, privacy, interoperability and availability to upgrade IoT devices over-the-air are some of the major challenges related to IoT.

Apart from them, there are a few challenges which are going to be very critical in the near future. These are *huge data volumes*, *real-time actions requirement*, *complex event processing* etc. With an industry estimate, billions of IoT devices are going to be connected every year (refer figure 1.1). They are estimated to be increased by CAGR[1] of 10% every year.

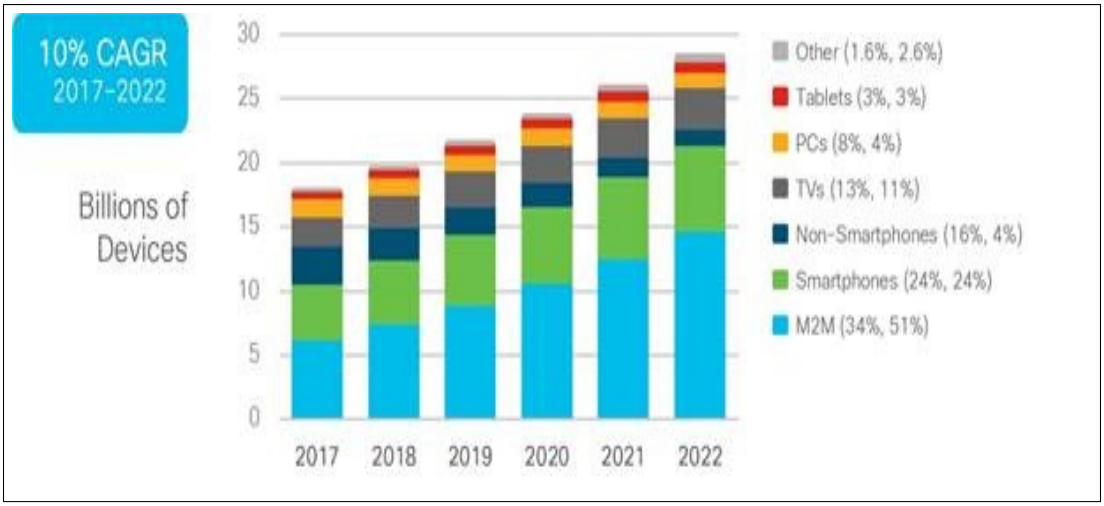

Figure 1.1 Connected devices' growth by 2022 [2]

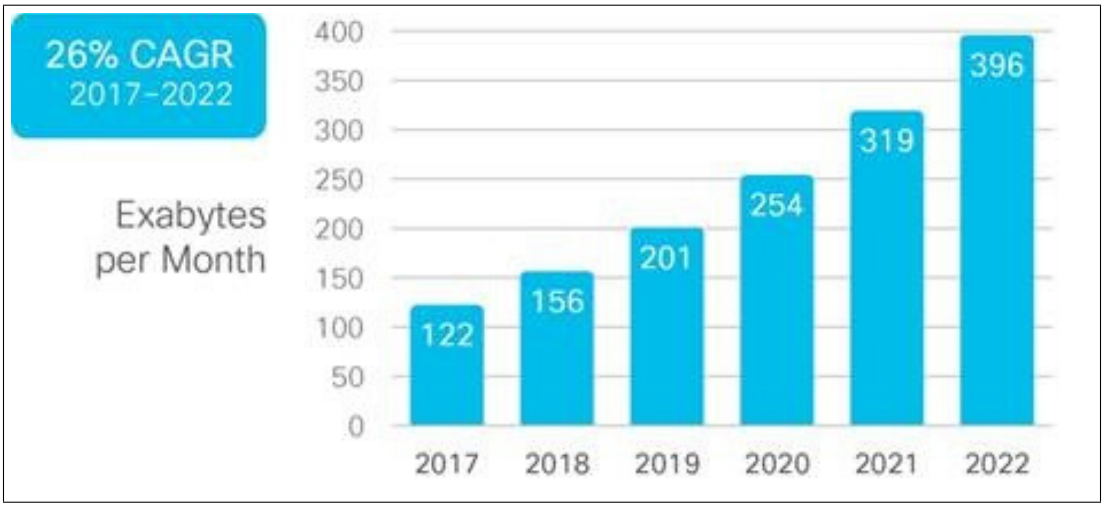

Figure 1.2 IP traffic (per month) forecast by Cisco by 2022 [2]

It is expected that all the connected devices are going to generate Exabytes[2] of data every month by the year 2022 (refer figure 1.2). It is expected to rise by a high CAGR of 26% every year till the year 2022. Considering this future scenario, for a limited number of IoT nodes, where strict delays are not a constraint, IoT-Cloud model would work fine. However, for quite a few real-time IoT applica-

[1] https://en.wikipedia.org/wiki/Compound_annual_growth_rate
[2] 1 EB = 1000^6 bytes = 10^{18} bytes

tions, this data requires near *real-time* complex event processing. Performing this task on Cloud can be catastrophic for such applications as it increases transmission delays involved in end to end processing. A few examples of real-time IoT applications are patient monitoring in ICU, automated cars, AR/VR, VANETs, real-time crowd surveillance etc.

In the centralized Cloud scenario, sensors and gateways have limited computing resources to generate insights from the sensed data. Most of the complex processing is done on Cloud and information/insight are been extracted to generate knowledge. Doing data processing on The Cloud may have some issues such as insufficient bandwidth, interrupted internet connection, high latency, security, delayed action and no real time insights etc.

The solution to this issue is to have decentralized processing close to the origin of the data. ***Edge computing*** is emerging as a solution to such scenarios [3]. The IoT application can survive using a local Edge server even if there is any fault in core internet connectivity. After processing the data on the edge, feedback can be sent to actuators in near real-time. After processing selected meta-data and insights are transferred to the centralized cloud. Edge servers are usually few metres or hop(s) away from the sensors. This resolves all the data processing issues discussed above such as insufficient bandwidth, delayed actions, intermittent connection etc.

1.1 Motivation

The motivation behind this research work is the following questions related to *reliable service* to real-time IoT applications:

What if due to the network or Cloud fault, IoT device is unable to connect to the cloud server?

What if any fault occurs on edge levels, either on middle edge or on extreme edge?

How to reliably assign edge resources to the IoT applications in a dynamic scenario?

This piece of research work tries to solve all these foreseen issues which are the result of a solution called Edge Computing.

1.2 Thesis Contribution

This research work presents

A reliable hierarchical Cloud architecture to manage Cloud failures and/or communication failures between IoT device and Cloud.

A reliable and fault-tolerant IoT-Edge architecture to manage Edge server failures with connection switching and efficient data store-forward mechanism.

An Edge Controller (EC) to assign Edge resources to the IoT applications reliably in a dynamic scenario.

Implementation of Reliable Edge in Landslide Early Warning System as an application.

1.3 Thesis Outline

Chapter 2 presents a hierarchical Edge Computing architecture as an advancement of the current IoT-Cloud architecture. It describes how Edge Computing can benefit the IoT applications if the central server or communication link to the server fails. This chapter also formulates and solve cost optimization problem for Edge Servers' deployment and optimal channel allocation in 5G's small cell communication networks.

In chapter 3, a reliable and fault-tolerant IoT Edge architecture is proposed and described. This chapter includes a novel agent-based system to keep the system running in case of Edge Servers' failures. It also demonstrates a mechanism to preserve data using multiple queues on the Edge Server(s) if the Cloud is unreachable. Further, analytical and simulation-based results are displayed in this chapter.

Chapter 4 proposes and describes a way to utilize privately owned and dedicated edge devices to meet the demand for real-time IoT applications. It is realized using an Edge Controller that can allocate edge resources reliably as per the delay tolerance of the applications. This chapter exemplifies and simulate various scenarios, formulate and optimize resource allocation by minimizing total turnaround time for all the tasks keeping reliability into consideration.

To realize the proposed mechanisms in this thesis especially in chapters 2 and 3 , chapter 5 displays the implementation of Reliable Edge for Landslide Early Warning System as an IoT application. This chapter applies the concepts of deploying an Edge Server between the Cloud and the Coordinator node on landslide location. Further, 2 alternate edge nodes are deployed as the part of the LEWS to demonstrate the scenario where the system can survive from a potential Edge failure if an alternate Edge Server is available.

Chapter 2

Edge Computing - A Hierarchical Architecture

2.1 IoT-Cloud Paradigm

Distributed Computing paradigms like Cluster Computing, Grid Computing were innovated to meet the increasing computing need of various applications. The culmination of such efforts is Cloud Computing [4] paradigm enabled by data centres, distributed across the world and the internet. It is commercially very successful.

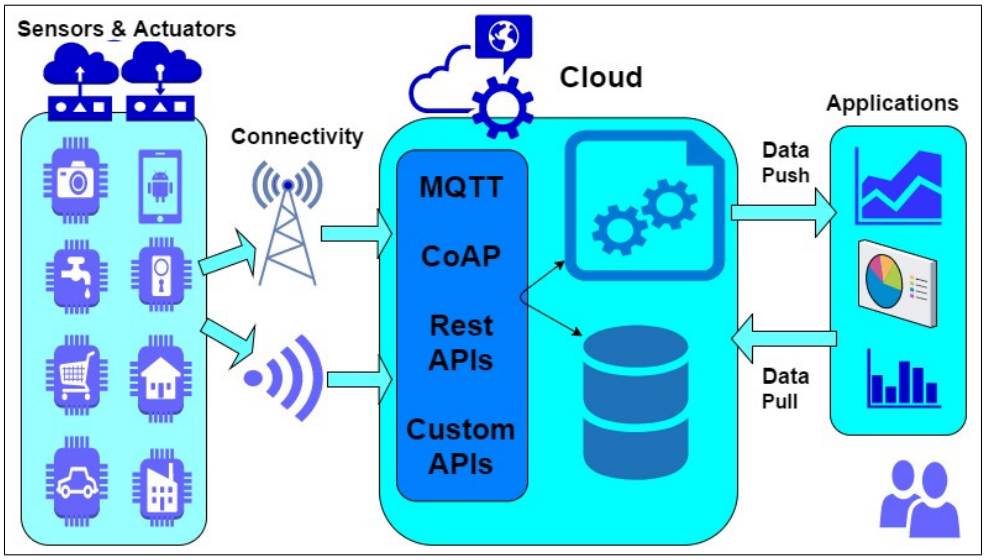

Figure 2.1 IoT-Cloud Infrastructure

Presently, IoT devices are connected to the Cloud through some (wired/wireless) communication medium. IoT-Cloud infrastructure is shown in figure 2.1. Sensed data reaches to the Cloud via core-internet for processing. This leads to a high communication cost (in terms of time delay), which can be catastrophic for many hard real-time delay-sensitive applications such as health monitoring, automated vehicles, manufacturing systems/assembly lines, video surveillance etc (figure 2.2).

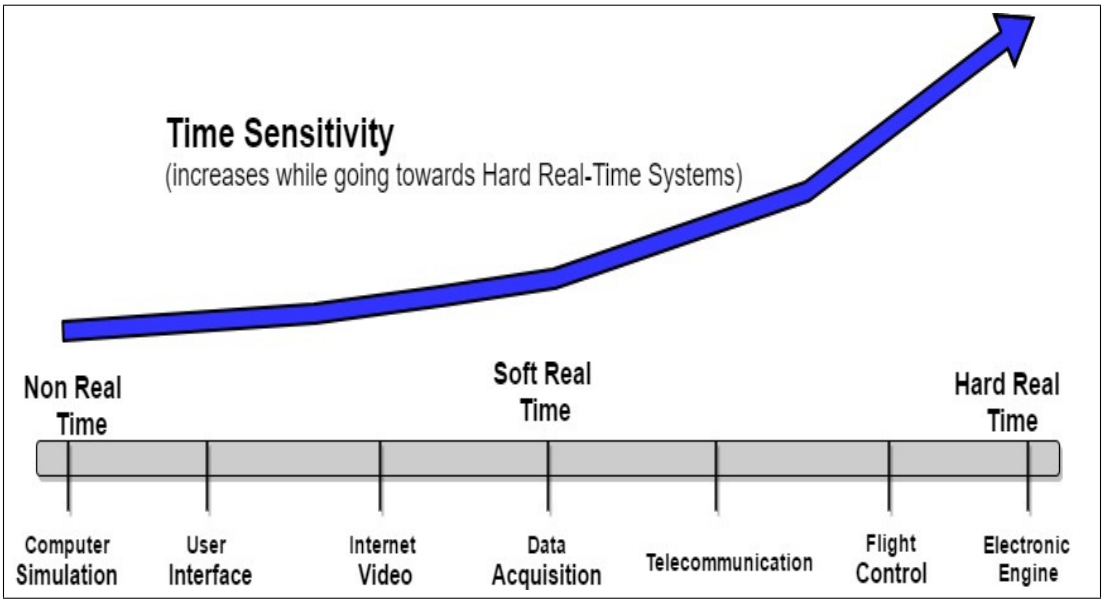

Figure 2.2 Time Sensitivity in Real-time vs Non-Real-time Systems

2.2 Edge Computing

2.2.1 Hierarchical Edge : Fog - Mist - Dew

It was realized that in context of technologies like IoT and CPS, the communication delay involved in processing data using the Cloud resources will be high and it is necessary to perform computing, communication and control (actuation) locally, near to the source of the sensed data. It is leading to the idea of Edge Computing [5].

The amount of communication delay between the IoT devices and the Cloud, and the load on the Cloud and core communication network during data transmission gave birth to the idea of computation on the edge of the network [6][7]. Variations of Edge Computing led to paradigms such as Fog, Mist and Dew Computing as shown in figure 2.3. It can be considered as a hierarchical Cloud where the Dew has the least latency and processing power, and the Cloud has the highest latency and processing power.

Extreme Edge is called ***Dew***. These are actually the end IoT devices. On Dew, the delay is almost negligible as there is no communication required with any other node to take any decision. Dew computing is used when the application requires real-time decision making without any delay. It works well if the requirement of computation power and previously sensed data is extremely low. In such cases, Dew Computing is very effective and saves energy, bandwidth etc. An automatic honk system on mountains for the drivers on U-turns is one of the examples of Dew Computing.

Mist (Roof) Computing also exists at the extreme edge of the network [8]. In Mist Computing, a server is established very near to the IoT device. For a sensing region such as an institute, it is installed usually in the same building and connected via a wired/wireless network. It is appropriate when the processing power and the previously sensed data's requirement is more than the capability of the Dew

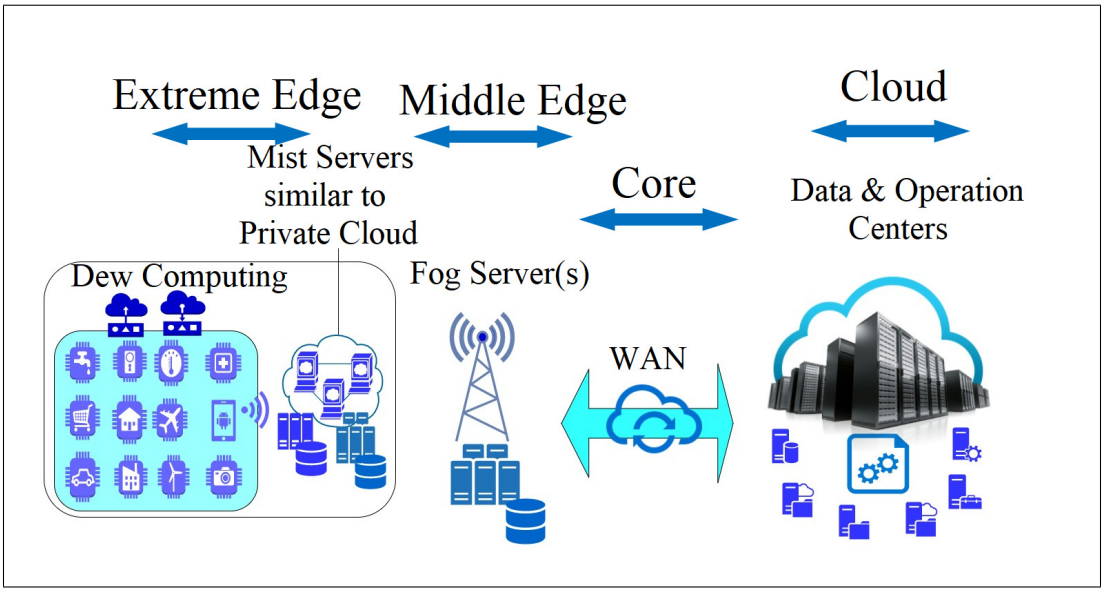

Figure 2.3 Hierarchical Edge-Cloud Architecture

level. The delay is also very low in Mist. The communication distance varies from a few meters to a few hundred meters.

Middle edge or ***Fog*** connects the extreme edge to the Cloud via the core network (WAN) [7]. Fog can be as distant as a few kilometres from the IoT devices [9]. The Fog Server can exist with the BS of an ISP or separately in the network. The computation power and the storage is higher in a Fog Server as compared to the Mist. There can be many Mist connected to one Fog. The distance of the Fog from the IoT devices is more than the Dew/Mist but it is still preferred due to its higher processing power and least delay as compared to Mist and Dew [10].

The Core is the WAN, comprised of all the traditional network elements like routers, bridges etc. If the Cloud is considered at the top, then it creates a hierarchy of networks with Fog, Mist and Dew on the lower levels.

2.2.2 Edge Computing for Reliability

Edge Computing potentially addresses issues like latency concerns, limited processing/storage capabilities of devices/things, battery life, network bandwidth constraints, security and privacy concerns etc. Edge computing [10] paradigm enables allocation of computing resources to the tasks generated by IoT applications depending upon their delay constraints.

Apart from the above mentioned issues, Edge Computing is capable of addressing one more issue that is *reliability* in the case of unavailability of a Central Server. This is one of such issues related to the Central Servers which hasn't been taken into consideration during the evolution of Edge Computing in last few years. Undoubtedly, Edge Computing and the Hierarchical Edge Architecture enables reliability feature in the traditional IoT-Cloud architecture.

Although, the probability of failure of a Cloud Server is negligible, there are IoT setups where servers are not as reliable as a Cloud data-centre with backups for computation resources, stored data, internet connectivity and power supply etc. So, there are possibilities of Central Servers' failures in such cases. Apart from that the bigger factor behind the unavailability of the Cloud Server or a Central Server is the network failure. Internet connectivity is one of the persistent problems in most of the developing countries. A few IoT devices installed in rural areas, mountains or flood areas etc mostly face this issue. Thus, irrespective of the reason, Edge Server(s) are a feasible alternate to the Central Server in the case of a failure. Edge Server can be connected through a P2P wired/wireless connection with sensors/CHs . These connections are highly reliable due to their adhoc nature and short range.

So, as far as reliability is concerned after the Central Server goes unavailable, an Edge Server

- may not be a true alternative of the Cloud Server, but certainly holds the potential to be a lighter version of it with lesser processing power and storage capacity.

- may not be able to save sensed data for many years, but it can save some data in the case of the unavailability of the Cloud and this data can be synchronized with the Cloud later on.

- may not be able to provide global access for long-term trends of data or responses based on years of historic data, but it can generate quick responses locally based on the small recent data-set available on the Edge.

Hence, Edge Computing is capable of converting the IoT-Cloud paradigm into a reliable one which can be really helpful for the applications where server's unavailability can be catastrophic.

2.2.3 Edge Communication

It is most likely that the edge wireless network is based on Micro/Pico/Femto cell infrastructure (figure 2.4). A multi-tier communication network usually overlays multiple tiers of cells and potentially share a common spectrum. A Macro-cell is usually a traditional cell tower which covers 15-30 kms but small cells cover a smaller area with respect to Macro-cell. A Micro-cell covers 1-2 kms. They can be used temporarily during the large events, or in heavy rush areas for additional coverage. Pico-cell can manage up to 100 users and covers 250 meters. They are mostly installed indoor and used to improve the coverage in an office or shopping area. Femto-cells are often self-installed and can manage a few users only. Small cells increase frequency reuse multiple times and enable cellular networks to manage a huge number of IoT devices on the edge. They can be installed at homes, offices etc to provide seamless connectivity to all the available IoT devices.

In the most interesting Edge Computing applications (such as in IoT), the packet traffic on the network has predictable patterns. Hence, Dynamic Channel Allocation (DCA) based on historical traffic data ensures optimal utilization of available channels. It is natural to employ cognitive radio technology to increase the spectrum utilization (as the demand for channels is sporadic by the devices). To facilitate

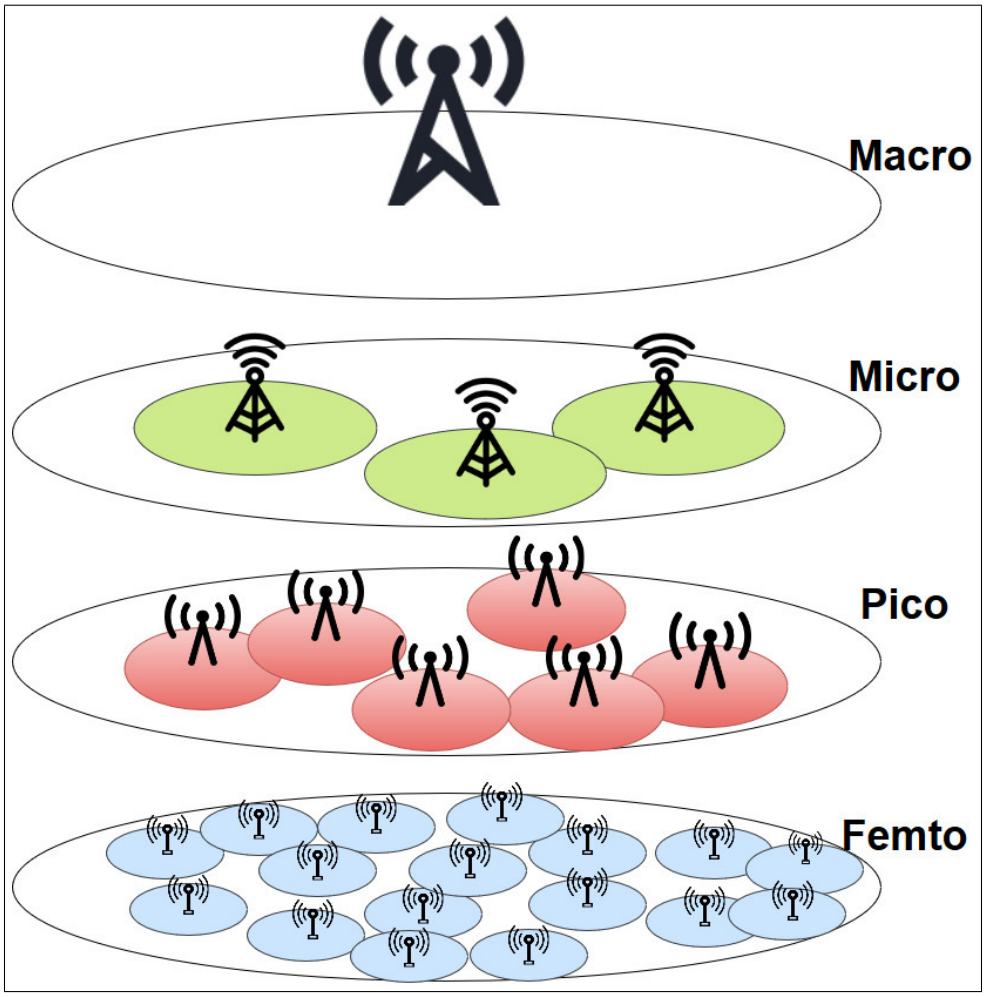

Figure 2.4 Multi-tire Communication Networks

cognition, wide-band spectrum sensing needs to be done in a time-optimal manner. Thus, it leads to a design of optimal cellular, cognitive radio network (Femto/Pico cell).

2.2.4 Edge Application

From the above discussion, the need for Edge Computing and Edge Communication is clearly evident. In this section, an application as an example which highlights finer requirements on the Edge Computing infrastructure.

Smart Hospital is one of the applications which require Edge Computing Infrastructure to deal with the delay sensitive patient monitoring system. It should be kept in mind that critically ill patients, effectively require real-time processing of the diagnostic data and real-time intervention by the doctor. Some regular patients generate diagnostic data that leads to predictable demands of edge resources

(computation/communication). But some type of data generated by patients in unpredictable. Hence, the Edge Computing paradigm needs to be flexible in provisioning edge resources based on demand.

In most hospitals, various types of data (example diagnostic data) with varied processing time constraints are generated. So, the data has features of Big Data. On the other hand, the storage of medical records of past and present patients needs to be done. The processing of such medical records to mine interesting patterns could be done using Cloud Computing resources. Thus, Cloud Computing and Edge Computing complement each other. Effectively real-time/non-real-time processing of patient data effectively aids medical diagnosis. The Edge Computing paradigm effectively has the following characteristics needed by various applications.

Upgradability of computing, communication, and control resources on demand.

QoS provisioning for various applications.

Security/Privacy of processed data etc.

It is crystal clear that there is a need for Edge Computing paradigms (such as Fog, Mist, Dew Computing). The resources at the edge are mainly computational resources (example multi-core processor based systems), communication/networking resources, control/actuation resources. These resources must be deployed and shared in an optimal manner to meet the QoS constraints of applications and next section focuses on it. Also, the cost of Edge Computing infrastructure needs to be minimized. [11] Since the connectivity structure of Edge Computing infrastructure is under the control of the user, the grid-based architecture provides a good approximation. Thus, it naturally leads to an interesting joint optimization problem with cost, delay etc as the objective functions, formulated further in this chapter.

2.3 Optimization in Edge Computing and Communication

In this chapter, an interesting optimization problems arising in the design of Edge Computing and Edge Communication network is formulated and solved witch is suitable for many applications. The motivation behind Edge Computing optimization problem is to minimize the cost of Edge Servers' deployment, taking location, type and cost of the servers into consideration. In a practical scenario, if the industry tries to provide an Edge Computing solution and wants to decrease the deployment cost, the proposed optimization solution can be used. The motivation behind small cell optimization problem is to allocate channels dynamically to meet the future demands of IoT devices. Use of Micro, Pico and even Femto cells is very much possible in the near future to provide seamless connectivity to IoT devices. Optimization of Multilevel Dynamic Channel Allocation (DCA) can help ISPs to serve more number of devices efficiently and provide better performance to end users.

2.3.1 Related Work

This section briefly describes the efforts made by the researchers to explain the need of Edge Computing. Further, the optimization strategies proposed for the Edge Servers' deployment would be discussed in this section. Amin et.al. [12] discussed the participatory Edge Computing for the local community services. They used and proved that using the local server is better if we have a complex core network available. They could successfully implement and provided third-party applications on local servers for the community near to them. Kim et.al. [13] worked with IoT devices to offload their computation on the Cloud. Based on the results, the authors found it better to offload and compute on the Cloud rather than doing computation on the IoT device itself. Samie et.al. [14] described a way of distributed computation offloading to many remote machines for distributed QoS for IoTs. It suggests decomposing the problem into small processes among many resources. Dynamic programming is used for resource allocation and the whole problem is solved using ILP. Mao et.al. [15] went one step further and offloaded the computation data to the Edge Server which is nearer as compared to the Cloud. By many examples it has been proved by the researchers that even though the machines on the edge have a lesser configuration, still, the delay is higher while getting the computation results from the Cloud. It happens due to the communication cost. In [16], the author proposed the optimizing solution by formulating it as an integer nonlinear program four offloading and resource allocation in Mobile-Edge Computing. Author found that due to the hardness of the problem, optimally solving it for the last scale network is impractical. To solve it efficiently the problem is divided into resource allocation and task offloading. Resource allocation problem is further divided into two sub-problems uplink power allocation and computing resource allocation. Convex and quasi-convex optimization techniques to solve these problems.

Now, this section briefly describes a few cases where researchers tried to optimize the Edge Server deployment strategy. Qiang Fan et. al. [17] gave a strategy CAPABLE to optimize the cost of server deployment and End to End delay between client devices and resources. The simulation results show that the scheme can trade-off between deployment cost and delay. Qin et.al. [18] have proposed a software-defined approach to manage heterogeneous IoT and sensor devices. This is done via providing the best matching resource for different classes of IoT devices. Authors have utilized reflective middleware with a layered IoT SDN controller for managing various IoT applications. It is the extension of multi-network information architecture (MINA). Farah Slim et. al. [19] proposes a mechanism related to the multi-dimensional Cloud. It gives an analytical model for blocking analysis. It also tried to find out the best strategy for distributed edge placement. Further, a strategy is devised for resource allocation and capacity planning on the edge network. So, the authors tried to optimize the limited resources available at the edge network.

For achieving energy efficiency taking feedback latency into account, mobile units demand a common/different Cloud Server for processing [20]. The proposed algorithm gives good performance for IoT with multi-small cells edge-computing and MIMO. The problem is formulated as the minimization problem but doesn't use learning methods to optimize the device-edge combinations. The proposed framework also works for radio access point implemented in distributed and parallel manna with limited

signaling to the Cloud. It is compared with disjoint optimization algorithm and shows better results. In IoT-Edge Computing scenario, it is very much necessary to learn the best combinations over the period of time.

This section covers some of the related proposed solutions but it is very clear that very few researchers are focusing on methods for optimizing the allocation of edge computation and communication resource to the IoT devices for their applications. Further, in the chapter, optimal allocation of edge resources using Integer Linear Programming is shown.

2.3.2 Edge Deployment Optimization: Integer Linear Programming

Consider devices/things which are locally distributed in space with limited local computation/memory resources (for example nodes in the Wireless Sensor Network). They require high-end Edge Computing platforms to process data efficiently. Also, it is necessary to be able to schedule the computing tasks onto a relatively high-end Edge Computing servers. The goal is to minimize the cost as well as the delay in the processing of the tasks generated by the local devices/things. Now, detailed modelling assumptions are given below.

There are two types $\{TypeA, TypeB\}$ of Edge Computing devices with costs $\{C_A, C_B\}$ (model can easily be generalized to the finite number of types of high-end computing devices).

The devices/things are distributed in a rectangular grid (model can be generalized to an arbitrary graph connectivity of devices)

The objective is to minimize the total cost of high-end computing platforms (required to process the "delay sensitive" tasks submitted by the devices/things) while at the same time ensuring that the things distributed on the rectangular grid are able to schedule their tasks onto a certain minimum number of high-end Edge Computing platforms. We now formulate the optimization problem as a $\{0, 1\}$ (binary/integer) Linear Programming Problem.

Let the variable associated with the placement of high-end Edge Computing platforms on the rectangular grid be $\{a_{ij}, b_{ij}$ where $0 \quad i \quad M, 0 \quad j \quad N\}$ i.e.

$$a_{ij} = 1; \text{if Type A Edge Server is placed at (i, j) location on the rectangular grid} \\ = 0; \text{otherwise} \tag{2.1}$$

$$b_{ij} = 1; \text{if Type B Edge Server is placed at (i, j) location on the rectangular grid} \\ = 0; \text{otherwise} \tag{2.2}$$

Let the grid points on the rectangular grid (where devices/things, edge computers are located) i.e. (M)(N) points be divided into mutually disjoint sets $\{D_\gamma : 1 \quad \gamma \quad L\}$ i.e. those sets constitute a set partition of all points on the rectangular grid. The constraint is that the points in each set D_γ (for all j) are served by at least "S" high-end Edge Computing platforms. Thus, the optimization problem has Linear Objective Function

$$\sum_{i=1}^{M}\sum_{j=1}^{N} C_A a_{ij} + \sum_{i=1}^{M}\sum_{j=1}^{N} C_B b_{ij} \tag{2.3}$$

subject to the linear constraints

$$\sum_{(i,j)2D_K} a_{ij} + \sum_{(i,j)2D_K} b_{ij} \quad S, \; for\, 1 \quad K \quad L \tag{2.4}$$

This Integer $(\{0, 1\})$ Linear Programming problem is solved using well-known techniques. Efficient algorithms exist for solving the problem.

An alternate formulation requires that every set D_δ (in the set partition of (M)(N) grid points) is covered by at least S_1, type A Edge Servers and by at least S_2, type B Edge Servers. Thus, the constraints in the above Linear Programming problem (Eq. 2.4) get modified in the following manner

$$\left\{ \begin{array}{ll} \sum_{(i,j)2D_K} a_{ij} & S_1; \\ \sum_{(i,j)2D_K} b_{ij} & S_2; \end{array} \right\} for\, 1 \quad K \quad L \tag{2.5}$$

2.3.3 Optimal Edge Communication Network: Integer Linear Programming

It is expected that small cells (Pico/Femto), enabling efficient frequency reuse, will be the important innovation in the deployment of the 5G cellular network. We consider the case where the wireless communication network which is providing connectivity between IoT devices/things at the edge is infrastructure based.

The goal is to make efficient utilization of channels available in the Pico/Femto cell. Thus, it naturally leads to DCA based on historical traffic data (at the edge) in adjacent small cells. Further, a dynamic spectrum access scheme based on time optimal spectrum sensing (using CR approach) is proposed. These approaches are formulated as optimization problems. Reasoning below demonstrates that these two optimization problems are related. In [21], the time-optimal spectrum sensing problem was formulated and solved. The solution is utilized for the efficient spectrum sensing in small cells. Further, the DCA problem in adjacent small cells is formulated and a solution is proposed.

Let there be M adjacent cells. Also, let the historical traffic (on some time unit example hour) in the spectrum bands in those cells be $\{n_1, n_2, n_3, ..., n_M\}$. $n_i^{\emptyset}s$ are normalized to arrive at Probability Mass Function i.e.

$$p_i = \frac{n_i}{\sum_{j=1}^{M} n_j}; for 1 \quad i \quad M \tag{2.6}$$

IDEA: In those bands where 'p_i' is small, allocate a small number of channels i.e. $n_i^{'}s$. On the contrary if p_i is allocated a higher number of channels in such a way that

$$\sum_{i=1}^{M} n_i.p_i = E[Z] \tag{2.7}$$

subject to the constraint that

$$\sum_{i=1}^{M} n_i = L \tag{2.8}$$

i.e. an average number of channels allocated to a small cell is maximized. It should be noted that 'Z' is the random variable associated with the number of channels allocated per cell.

It is reasoned below that, if there are no constraints imposed on $n_i^{'}s$, the problem becomes trivial.

Problem formulation: Maximize $E[Z]$

Subject to the constraint that

$$\sum_{i=1}^{M} n_i = L \tag{2.9}$$

Where L is the total number of available channels.

For instance, the probabilities can be ordered as

$$q_1 \quad q_2 \quad \dots \quad q_M \tag{2.10}$$

Set $n_M = L$ and $n_i = 0$ for $i \neq M$. Thus, L is the maximum attainable value for $E[Z]$. Thus, it naturally leads to impose reasonable constraints on $n_i^{'}s$ motivated by practical considerations.

$n_i^{'}s$ are in A.P. i.e. $n_1, n_2 = n_1 + d, n_3 = n_1 + 2d, ..., n_M = n_1 + (M - 1)d$

From the point of view of the allocation of channels, these constraints are very reasonable and implementable.

Now, the above precise optimization problem is solved. The case is considered where the number of channels allocated per cell are in Arithmetic Progression. Thus, the constraint leads to

$$n_1 + (n_1 + d) + ... + (n_1 + (M - 1)d) = L \tag{2.11}$$

Since $\{n_1, d\}$ are integers, it leads to the Linear Diophantine equation

$$2Mn_1 + d(M)(M - 1) = 2L \tag{2.12}$$

Thus, the solutions of the Eq. (2.12) is utilized to solve the stochastic optimization problem of maximizing $E[Z]$.

First the probabilities are sorted i.e. $p_i^{'}s$ in increasing order resulting in relabelled probabilities $q_i^{'}s$ that is

$$q_1 \quad q_2 \quad \dots \quad q_M \tag{2.13}$$

Thus, maximization of $E[Z]$ requires that

$$\widetilde{n}_1 \quad \widetilde{n}_2 \quad ... \quad \widetilde{n}_M \tag{2.14}$$

It can be readily seen that

$$E[Z] = \widetilde{n}_1 + (\delta)d; where\ \delta = \sum_{j=1}^{M}(j \quad 1)q_j \tag{2.15}$$

Also, computing variance of Z, it is

$$Var[Z] = (\alpha \quad \delta^2)d^2; where\ \alpha = \sum_{j=1}^{M}(j \quad 1)^2 q_j \tag{2.16}$$

The following theorem provides a unique solution to the above optimization problem.

Theorem: Assume that $f(a_1, d_1), ..., (a_l, d_l), ..., (a_k, d_k)g$ are the set of solutions of Linear Diophantine Eq. (2.12) among the infinitely many solutions, which are positive real integers. If $a_1 < ... < a_l < ... < a_k$ then $d_1 > ... > d_l > ... > d_k$. In such a case, (a_1, d_1) is the best solution, which maximizes the expected value $E(Y)$.

Proof: To prove that (a_1, d_1) maximizes $E(Y)$, expression in equation 2.17 is required to be proved.

$$a_l + \delta d_l \quad a_1 + \delta d_1 \quad for\ l = 2, 3, ..., K \tag{2.17}$$

$$i.e.\ \ a_l + \delta d_l \quad a_1 + \delta d_1 \quad for\ l = 2, 3, ..., K \tag{2.18}$$

$$i.e.\ \frac{a_l \quad a_1}{d_1 \quad d_l} \quad \delta \quad for\ l = 2, 3, ..., K \tag{2.19}$$

Since, (a_1, d_1) and (a_l, d_l) both satisfy the Linear Diophantine equation (2.12). Therefore, it can be written as

$$M \quad a_1 + \frac{(M \quad 1)M}{2} \quad d_1 = L \tag{2.20}$$

$$M \quad a_1 = L \quad \frac{(M \quad 1)M}{2} \quad d_1 \tag{2.21}$$

$$M \quad a_l + \frac{(M \quad 1)M}{2} \quad d_l = L \tag{2.22}$$

$$M \quad a_l = L \quad \frac{(M \quad 1)M}{2} \quad d_l \tag{2.23}$$

Multiplying Eq. (2.19) by M on both sides, it becomes

$$M \quad \frac{a_l \quad a_1}{d_1 \quad d_l} \quad M \quad \delta \tag{2.24}$$

$$\frac{M \quad a_l \quad M \quad a_1}{(d_1 \quad d_l)} \quad M \quad \delta \tag{2.25}$$

$$\frac{[L \quad \frac{(M \quad 1)M}{2} \quad d_l] \quad [L \quad \frac{(M \quad 1)M}{2} \quad d_1]}{(d_1 \quad d_l)} \quad M \quad \delta \tag{2.26}$$

$$\frac{\frac{(M \quad 1)M}{2} \quad (d_1 \quad d_l)}{(d_1 \quad d_l)} \quad M \quad \delta \tag{2.27}$$

$$\frac{(M \quad 1)}{2} \quad \delta \tag{2.28}$$

Substituting the value of δ, the resultant is

$$\frac{(M \quad 1)}{2} \quad \sum_{j=1}^{M}((j \quad 1)p_j \tag{2.29}$$

Therefore, it is enough to prove the above expression in order to prove (a_1, d_1) is the best solution which maximizes $E(Y)$.

By the proof of contradiction, it can be proved that the equality in Eq. (2.29) holds only for uniform distribution, i.e. $p_1 = p_2 = ... = p_M = \frac{1}{M}$. Therefore,

$$\sum_{j=1}^{M}(j \quad 1)p_j = \frac{1}{M}\sum_{j=1}^{M}(j \quad 1) = \frac{(M \quad 1)}{2} \tag{2.30}$$

For any other probability distribution, the equation would be

$$\frac{(M \quad 1)}{2} < \sum_{j=1}^{M}(j \quad 1)p_j \tag{2.31}$$

Proof: To prove the minimum value of $\sum_{j=1}^{M}(j \quad 1)p_j$ occurs only for uniform distribution, i.e., $p_1 = p_2 = ... = p_M = \frac{1}{M}$, proof through contradiction is used. Suppose that the probability masses $p_1 = q \quad \epsilon, p_2 = p_3 = ... = p_{M\ 1} = q = \frac{1}{M}$ and $p_M = q + \epsilon$ gives the minimum value of $\sum_{j=1}^{M}(j \quad 1)p_j$, where $\epsilon > 0$ is a small positive real number. Using the given probabilities, value of the expression $\sum_{j=1}^{M}(j \quad 1)p_j$ can be calculated as given below:

$$\begin{aligned}\sum_{j=1}^{M}(j \quad 1)p_j = 0 \quad (q \quad \epsilon) + \sum_{j=1}^{M}(j \quad 1) \quad q \\ + (M \quad 1) \quad (q + \epsilon)\end{aligned} \tag{2.32}$$

$$\begin{aligned}\sum_{j=1}^{M}(j \quad 1)p_j &= \sum_{j=1}^{M}(j \quad 1) \quad q + (M \quad 1) \quad (q + \epsilon) \\ &= \sum_{j=1}^{M}(j \quad 1) \quad q + (M \quad 1) \quad \epsilon \\ &= \frac{1}{M} \quad \sum_{j=1}^{M}(j \quad 1) + (M \quad 1) \quad \epsilon \\ &= \frac{1}{M} \quad \frac{(M \quad 1)M}{2} + (M \quad 1) \quad \epsilon \\ &= \frac{(M \quad 1)}{2} + (M \quad 1) \quad \epsilon\end{aligned} \tag{2.33}$$

Since, $\epsilon > 0$, it contradicts the supposition. Therefore, it can be concluded that the value of $\sum_{j=1}^{M}(j - 1)p_j > \frac{(M-1)}{2}$ for any probability distribution and minimum value occurs for uniform distribution.

2.4 Summary

In this chapter, a hierarchical Edge-Cloud architecture as an extension of traditional IoT-Cloud architecture has been demonstrated. It is valuable in many ways such as near real-time responses, energy efficiency, and data security etc. Additionally, it can be extremely beneficial to provide reliability on the case of unavailability of a Central Server due to any reason. Edge Server ensures that the IoT applications keeps on working in case of a server's failure. Furthermore, this chapter focuses on the cost-effective Edge Server placement problem, considering the location of the Edge Servers are grid points. This is solved using 2 types of Edge Servers that can easily be generalized to any number of servers' type. Along with this, the issue of optimal spectrum sensing and optimal dynamic channel allocation problem arising in small cells based on historical traffic data is solved.

Chapter 3

Reliable and Fault-Tolerant Edge

Edge Computing [22] and Hierarchical-Cloud Architecture proposed in the last chapter seems to solve the issue of delay, bandwidth and fault on Cloud level, but what if any fault occurs on edge levels, either on middle edge or on extreme edge? In this chapter, a reliable and fault-tolerant architecture for the IoT-Cloud hierarchy is proposed.

3.1 Related Work

Many researchers have proposed solutions for fault tolerance in IoT-Cloud infrastructure. Some of the proposed solutions are given in this section. A framework for providing a hybrid fault tolerance in Cloud computing [23] [24] [25] addressed the issues of fault tolerance that can cause the failure of the Cloud.

In [26], the authors proposed a distributed Cloud storage architecture in which services or data is provided to the user in a manner that they can access it locally without having an internet connection. The proposed architectures do not provide any solution to handle replicated data stored on users machines. In [27], an algorithm is proposed to increase the reliability of the system by using virtual machines. Virtual machines adapt changes of the system in every cycle and generate correct results. The drawback of the proposed algorithm is that there is no schedule management system or a decision-making system.

In [28], the authors proposed a fault tolerance session layer to overcome the drawback of event-driven communication. With the help of adding new layers and checkpoints, it maintains fault tolerance. This mechanism uses a large number of threads that may reduce the performance of the server. A fault tolerance framework based on the artificial neural network to detect faults and maintains a gap between Cloud Servers and users is proposed in [29]. In [30][31], the authors presented an evaluation study on different existing solutions for managing efficient Cloud storage and utilization of resources with different quality assurance parameters. They proposed a scheduling mechanism in which user requests and Clouds services are managed or scheduled efficiently but those algorithms work only with the centralized environment but fails in a distributed environment. Also, the proposed framework works based on static information of resources and users requests.

Leveraging Fog computing for scalable IoT data-centre, a mechanism to overcome latency and bandwidth issues is proposed in [32]. In this mechanism, the concept of spin leaf helps to manage big data on Cloud by offload and batch processing, but it does not support multi-node architecture. Also, it is not so efficient for scheduling related problems.

3.2 Reliability, Fault Tolerance and Fault Localization

With the increase of IoT devices in the system, maximization of the computation on edge network is required to minimize the communication cost. Self-managing and self-configuring solutions are also required on edge network. The IoT application must be able to recover from any issues that arise during its lifetime. Mist and Fog Computing should offer these features. So, in some sense, it has been believed that Mist and Fog will address many of the challenges that are being faced in dealing with large-scale IoT systems. Now, if this hierarchical architecture is adopted, some of the biggest concerns are reliability and fault tolerance. There are several concerns listed below related to reliability and fault-tolerance that need to be taken care of.

The whole system architecture should be able to provide services even if any node (server) fails on any level.

The sensed data should be replicated and available on other parallel nodes to take over the control in the case of a node failure.

There should be an application interface available on the newly assigned alternate server. It should be managed virtually without any inconvenience and knowledge to end user.

The switching from one server to another should be automatic, and it could be based on priority. For example, if one Fog fails then its IoT applications should be shifted to another Fog, Mist or to Cloud (in the worst case), automatically. The decision depends upon the delay constraints of the applications.

The new servers reconfiguration and path redirection time should be minimal.

3.3 Proposed Architecture

3.3.1 Reliability and Backup Policy

To construct a reliable IoT-Cloud infrastructure, the replication of sensed data is very important. The sensed data's redundancy ensures the feedback for the actuators in case of any server-side failure. [33][34] The backup system can take over the control in such cases. For Clouds, the mechanism of AZ (Availability Zone) for the backup is used in case of any disaster or failure in the Cloud Server.

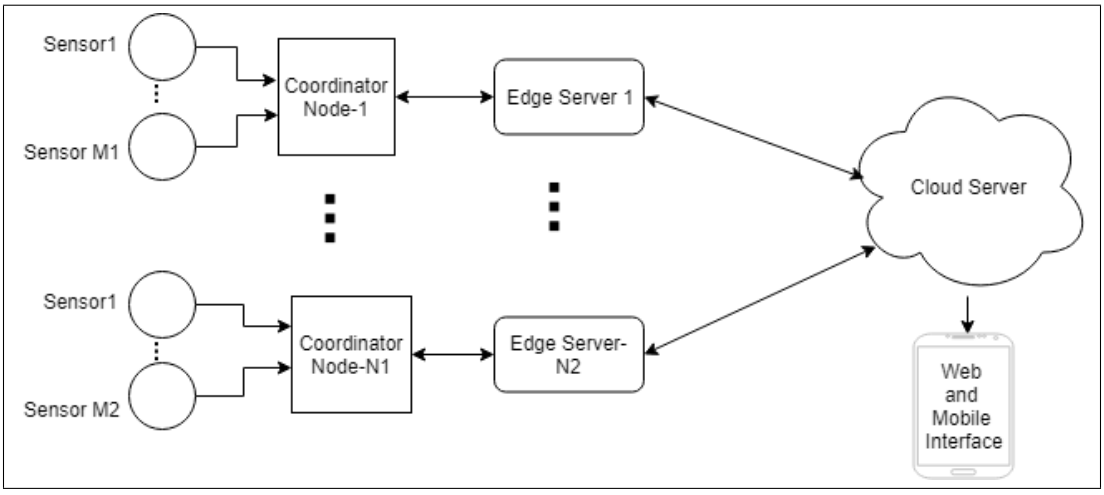

Figure 3.1 IoT-Cloud Architecture with Edge Computing

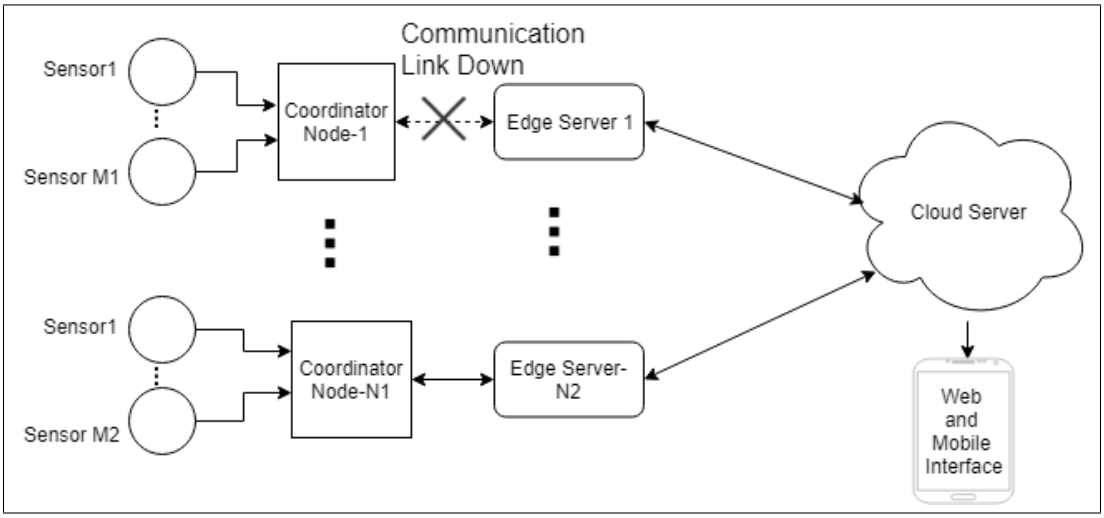

Figure 3.2 Edge Failure or Communication Link Failure between Coordinator Node and Edge Server

Now, while using the concept of Edge Computing, the focus is on running IoT applications from the edge of the network rather than from the far Cloud. Figure 3.1 depicts the IoT-Cloud architecture in reference with Edge Computing. There are one or more sensors connected to each of the N1 Coordinator Nodes and these Coordinator Nodes are further connected to N2 number of Edge Servers. More than one Coordinators can be connected to each of the Edge Servers. All the Edge Servers are further connected to a Cloud Server to complete the hierarchy. Edge level servers should also have backup sibling servers to keep the applications running in case of any Edge Servers failure as shown in figure 3.2. Figure 3.2 shows that if system does not have any alternate backup plan after any edge failure then the respective zone can face a serious outage. So, to prevent the system from such failures, an alternate support scheme has been presented as shown in figure 3.3. It shows that if any Edge Server (such as Edge Server-1) goes down then its associated Coordinator Node-1 will look for any nearby available Edge Server. If any

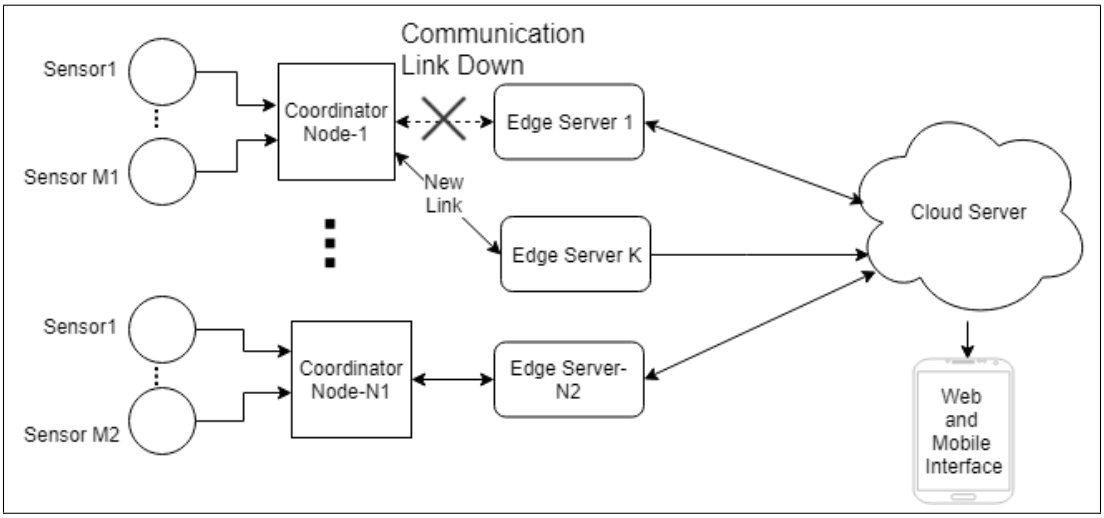

Figure 3.3 Communication Link Re-establishment between the Coordinator Node and an Alternate Edge Server

match is available (such as Edge Server-K in the figure) then this Edge Server will work as an alternate Edge Server for the Coordinator Node-1. Alternate Edge Server concept will be used at the time of any outage. With any outage of Edge Server, there can be only one associated alternate Edge Server with the Coordinator Node. Coordinator Node has inbuilt search mechanism to find an alternate Edge Server if its own Edge Server is unavailable. The alternate Edge Server then takes the responsibility to provide services to the IoT application and to keep the communication continue. This is going to be an automatic virtual process. The proposed multi-level architecture is reliable in the sense that even if there are no alternative servers on the same level with sensed data replication, there is a guaranty of replication on the higher level (refer figure 3.3). The volume of data for the same application may vary on Mist, Fog and Cloud levels. But data would exist on all the levels to take over the control of the application in the case of a servers failure.

3.3.2 Fault Tolerance and Agent

By definition, Mobile Agent (MA) is a special purpose software code that can transfer itself from one machine to another. Practically, MA is the same code running in all the machines and transmits data from one machine to another. MAs are very helpful in managing distributed systems. In the proposed work, the faults in the entire hierarchy will be managed by MAs.

The MA works as a resource and network monitoring agent. It shares the application and link state information with other agents on alternative servers in the hierarchy. Except for monitoring, they are also responsible for assigning the priority to the IoT applications depending upon their delay-tolerance. In the case of impacted failure or impacted scheduled shutdown, this priority information is used at the time of load distribution [35] of a particular server. It also helps in new path discovery [36] after the

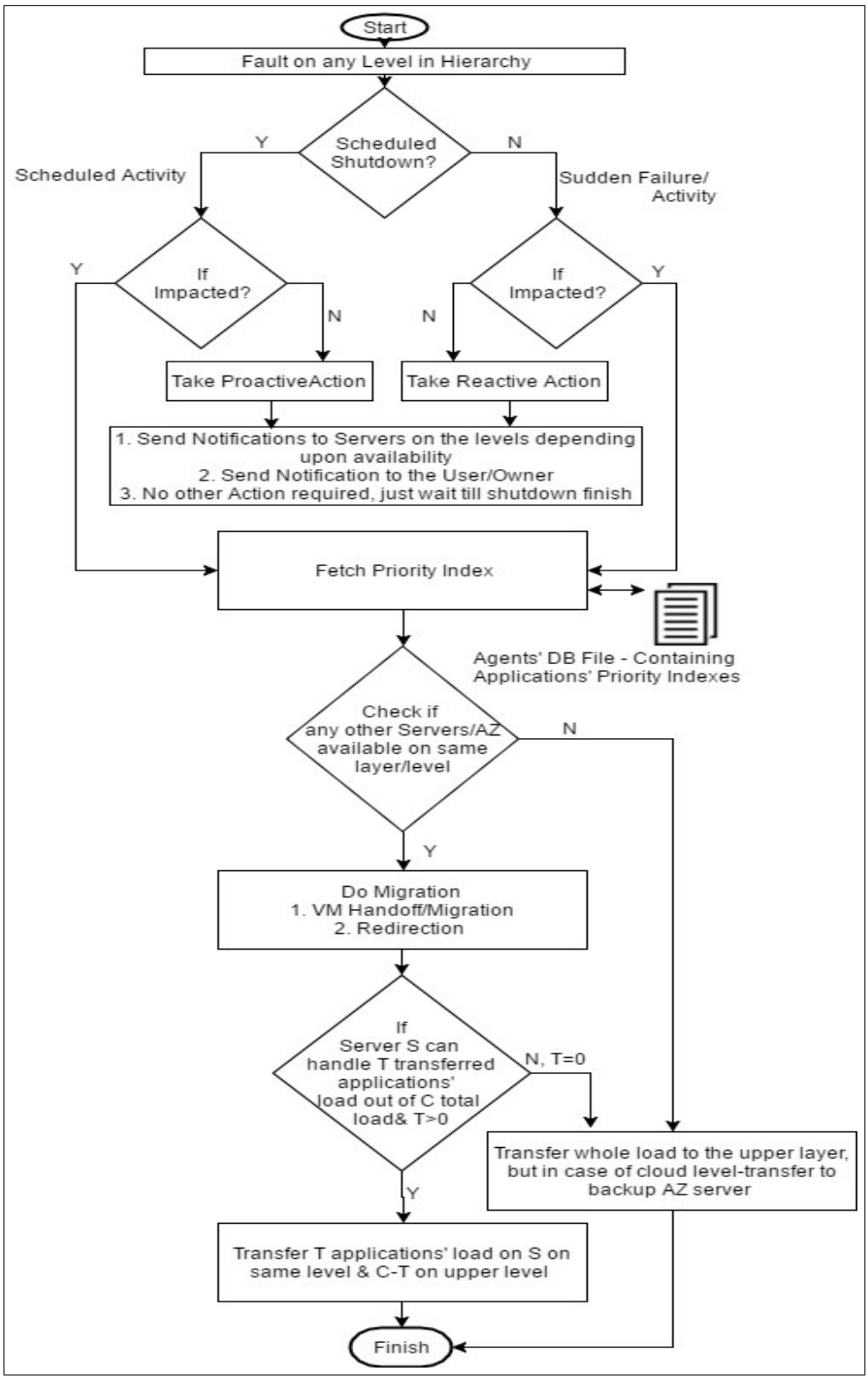

Figure 3.4 Agents Working during Faults

load distribution. It also keeps a check on the periodic monitoring and backup of data. Figure 3.4 shows the recovery process in case of faults and scheduled shutdowns, reactively and proactively respectively.

Figure 3.4 depicts the complete fault tolerance cycle of the hierarchy between Dew, Mist, Fog and Cloud. Here, the fault tolerance has been achieved by exploiting the capabilities and benefits of an agent, which is basically a software program running on each server. At the time of any fault, the whole responsibility is assigned to the same level agent via a higher level agent in the hierarchy. Agents working is given below:

As per assumption

- The reason behind any fault/shutdown can either be any scheduled activity or any sudden activity.
- Any sudden or scheduled activity comes with two possibilities that either it is going to affect respective server or no effect at all on it.
- If the ongoing activity has no effect then for a scheduled event, the agent will move with proactive action which is basically to send notifications to all respective agents, whereas for sudden activity agent will look for reactive action.

For any sudden fault, the agent will fetch priority index for all applications running on the affected server and it will immediately check if any other server is available on the same level via a higher-level node in the hierarchy. After that, it will do application migration and connection redirection and then will do the load transfer activity as per the sequence is given below:

Algorithm 1 Agents working on Edge

Total load = C

Load handling capacity of Server S = T

if $T > 0$ **then**

    S T (load)

    Upper level $(C$ $T)$ load

if $T = 0$ **then**

    Upper level Transfer whole load

If no other same level server is available then application(s) will connect to the upper level computing resource(s) to transfer the whole load.

3.4 Multi-queue Store and Forward Mechanism for Edge Server(s)

There is a possible scenario when Edge Servers are not connected to the Cloud Server. Yet, they are able to receive data from the Coordinator Nodes located at the Extreme Edge. This duration of unavail-

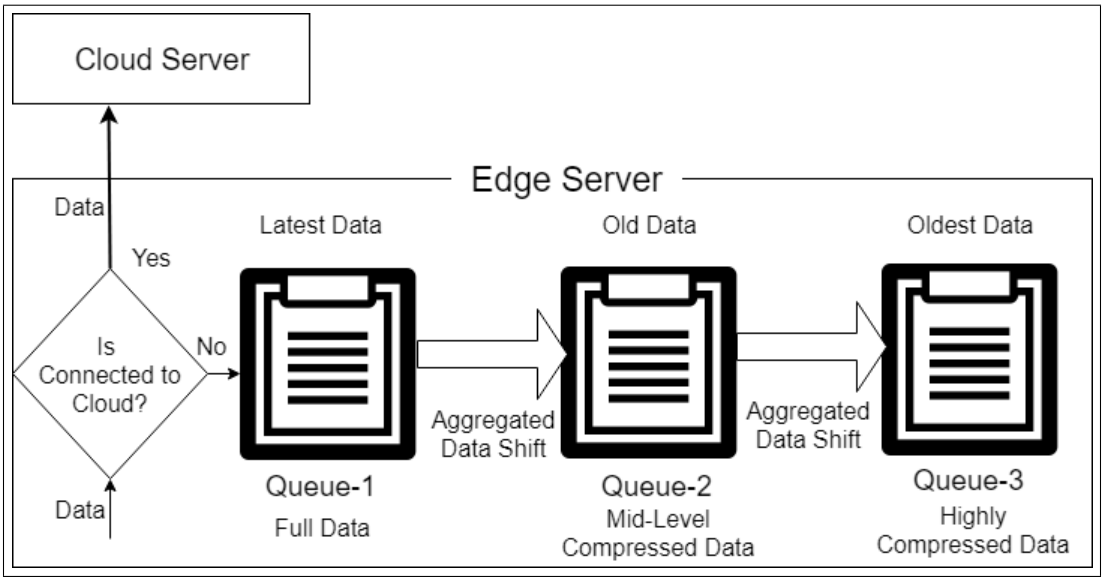

Figure 3.5 Multi-queue Storage Mechanism with Data Aggregation

ability of the Cloud may vary from a small delay in transmission to a long period of disconnection. In this scenario, a Reliable Edge Server needs to store the sensed data temporarily, but in an efficient manner as they have very limited storage space. To decrease the possibility of loosing sensed data during disconnection, an application of a multi-queue store and forward mechanism has been proposed here for a Reliable Edge.

In the store and forward mechanism, data is transferred between one device to another but it has to pass through an availability check. This availability check ensures that the data will remain in the queue until the receiver connects back and the data is transferred to the intended receiver afterwards. After transmitting the data, it is removed from the respective queue. This mechanism works fine when either there is no issue with the storage space on the sender node or the data loss is not an issue if the queue gets full. That is never the case with Edge Servers.

Due to the limited storage space on the Edge Servers, a single queue mechanism cannot work. If a single queue gets full then data needs to be discarded to save new data or new data needs to be dropped. A Reliable Edge cannot afford this. A better alternative is proposed here by using 3 queues simultaneously. The working of these queues is discussed below (refer figure 3.5):

All queues will work on First In First Out (FIFO) method. It means that the oldest data will pop out first.

It is assumed that the total storage capacity of the system is 100-units and it is divided into three parts that are 70%, 20% and 10% for Queue-1, Queue-2 and Queue-3 respectively. Given these conditions, all three queues work as follows:

- Queue-1: Queue-1 can consume 70% of the available storage space. This queue contains the uncompressed sensed data which means that the sensed data is not aggregated and is kept in the original form as it is supposed to be received by the Cloud.
- Queue-2: Queue-2 can consume 20% of the storage space for storing the older data. Queue-2 contains Level-1 compressed/aggregated data. It receives data only if the Queue-1 touches its 70% of the storage space or in other words it gets full. Queue-1 pops out its oldest data and aggregates the sensed data to compress and shifts to Queue-2. For example Queue-1 is receiving a sensor's data with frequency 24/day. Level-1 compression aggregates the data and converts it into frequency 6/day by combining 4 data values into 1 using the techniques like average, max or min etc.
- Queue-3: Queue-3 consumes 10% of the total storage space to store the oldest data and the most compressed one. Queue-3 will contain Level-2 compressed data. If the duration of disconnection goes really long and Edge Server needs to store the data for the entire period, Queue-3 is essential. If both Queue-1 and Queue-2 reach their thresholds, Queue-3 requires to store highly compressed and oldest data. For this, oldest data from Queue-2 pops out and aggregates to decrease the number of entries to store it in Queue-3. Following the example given for Queue-2, data can be compressed further to make it 1/day by aggregating 6 data values.

Once the Cloud Server (receiver) successfully connects back, the most recent data is forwarded first to the Cloud Server. After the transfer of most recent data from Queue-1, Queue-2 and Queue-3 data is synchronized with the Cloud Server and all the Queues get empty.

3.5 Analysis of Reliability and Availability in an Edge-based IoT Infrastructure

In order to create a reliable system, the availability of the alternate Edge Nodes is analyzed to offer the seamless transmission of the sensors data to the Cloud Servers. The proposed mechanism and architecture attempts to make a system that ensures the availability and reliability during the processing and transmission of the sensors' data. So, it can be said that the system runs without failure and unavailability of the intermediate devices and the Internet, respectively.

3.5.1 Availability

It is the ability to transfer the information or performed specified function even in case of failure of some edges/devices. The availability will be measured in terms of probability at any time which depends upon the Edge's up-time and down-time. The availability (A_t) is defined as:

$$Availability\ (A_t) = \frac{Uptime}{(Uptime\ +\ Downtime)} \tag{3.1}$$

3.5.2 Reliability

It is the ability of the system that consistently performs specified function even in case of failure of some Edge Nodes. The reliability will be measured in terms of probability at any time which depends upon the failure probability. Reliability is defined as:

$$Reliability\ (R) = 1 \quad Failure\ Probability\ (F_p) \tag{3.2}$$

The failure probability derives from the ratio of the number of failed trials and a total number of trials during the transmission of the sensor data to the Edge Nodes. Failure probability is defined as:

$$Failure\ Probability\ (F_p) = \frac{Number\ of\ Failed\ Trials}{Number\ of\ Trials} \tag{3.3}$$

3.5.3 Analysis of Reliability Performance

The failure frequency is used to measure the reliability of the proposed system.

Let us consider a scenario when a failure has occurred, the system pause and migrates the current tasks to the neighbour Edge Server and resume them for seamless execution and transmission of the information. The migration/switching time and tasks transfer time between Edges is neglected because of all these operations are completed in constant time. The number of redundant nodes increases data transfer availability and build a durable system. For example, simply using two alternate Edge Nodes provides the 0.99 availability.

Considering an example scenario where the total number of existing Edge Nodes are N = 1000 and M = 10% of them are currently unavailable with an approximate 23.5 hours of time Edge Nodes up. Maximum T = 20 numbers of trials allow checking the working status of any Edge Nodes. Then the average failure probability is 0.03 (= 0.6/20) with 3% failure trials. So, the probability of reliability is 0.97 (= 1-0.03). The probability of availability is 0.9791 (=23.5/23.5+0.5).

After considering multiple scenarios with different parameters to analyze the availability and reliability based on Uptime, Downtime, Number of Trials, and Number of Failed Trials, their Availability and Reliability matrix is given in table 3.1. Figure 3.6 and figure 3.7 displays the graphs based on the table. Figure 3.6 shows the comparison of reliability performances for a number of experiments and figure 3.7 displays the comparison graph of availability for the better understanding of the results produced after several experimentation cases.

3.6 Simulation Result and Analysis

Both agent-based and without-agent solutions are simulated using Matlab. Figure 3.8 shows the application assignment efficiency of the system during faults with MAs in comparison to random as-

Table 3.1 Availability and Reliability Matrix for various Cases

Experiment No.	Uptime	Downtime	No. of Trials (T)	Number of Failed Trials	Availability (A)	Failure probability (F_p)	Reliability (R)
1	22	2	17	3	0.91666667	0.17647059	0.82352941
2	22	2	18	2	0.91666667	0.11111111	0.88888889
3	22	2	19	1	0.91666667	0.05263158	0.94736842
4	23	1	17	3	0.95833333	0.17647059	0.82352941
5	23	1	18	2	0.95833333	0.11111111	0.88888889
6	23	1	19	1	0.95833333	0.05263158	0.94736842
7	23.05	0.95	17	3	0.96041667	0.17647059	0.82352941
8	23.05	0.95	18	2	0.96041667	0.11111111	0.88888889
9	23.05	0.95	19	1	0.96041667	0.05263158	0.94736842
10	23.08	0.92	100	3	0.96166667	0.03	0.97
11	23.08	0.92	100	2	0.96166667	0.02	0.98
12	23.09	0.91	100	1	0.96208333	0.01	0.99

signment. With the help of MAs, job distribution for the idle server is high in numbers and null for under maintenance servers. While job distribution for busy and backup servers performed efficiently. If the system proceeds without an agent then it distributed jobs randomly without acknowledging the servers state. Figure 3.9 shows the CPU time unit consumption with the increase of server counts. When the system proceeds with MAs then the performance is very high in spite of the the centralized server or the Edge Server goes unavailable while serving IoT applications.

3.7 Summary

The proposed work in this chapter puts efforts for improving fault-tolerance and reliability of Edge Computing with the help of an intelligent agent. The given novel architecture provides solutions for the possible faults at any level of the Cloud-hierarchy especially at the edge. To deal with any fault or issue, the proposed concept works with both types of solutions that are proactive and reactive. Results also show the efficiency of the proposed architecture. This is a practical solution and tested by implementing it for a suitable IoT application.

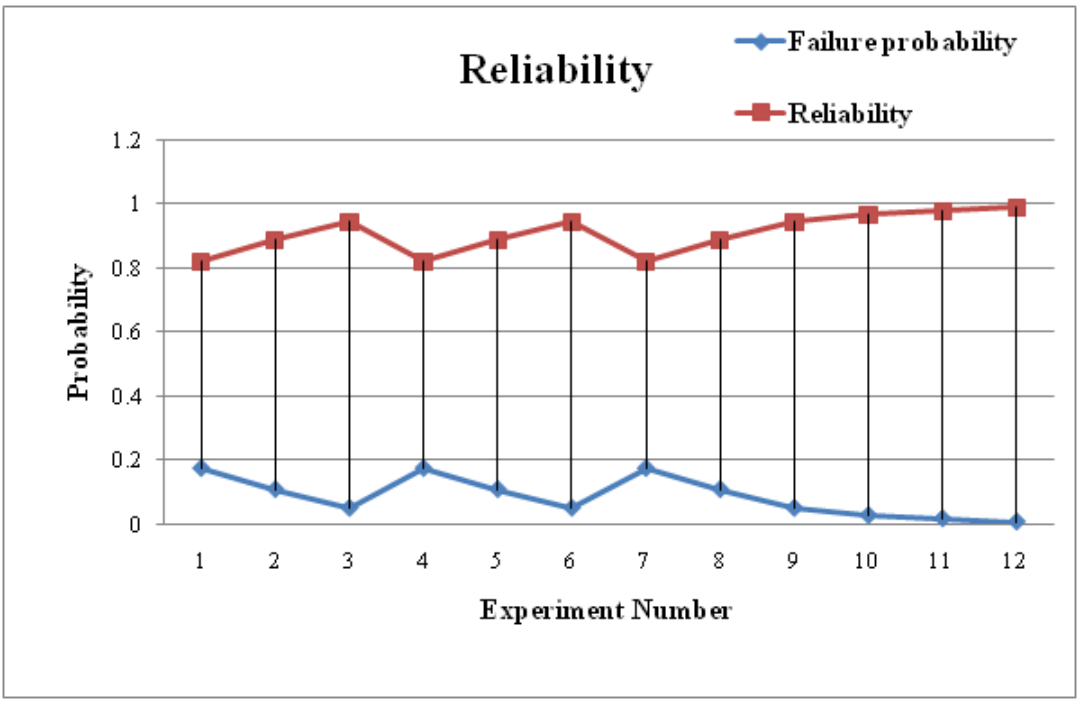

Figure 3.6 Comparison of Reliability Performances for a number of experiments

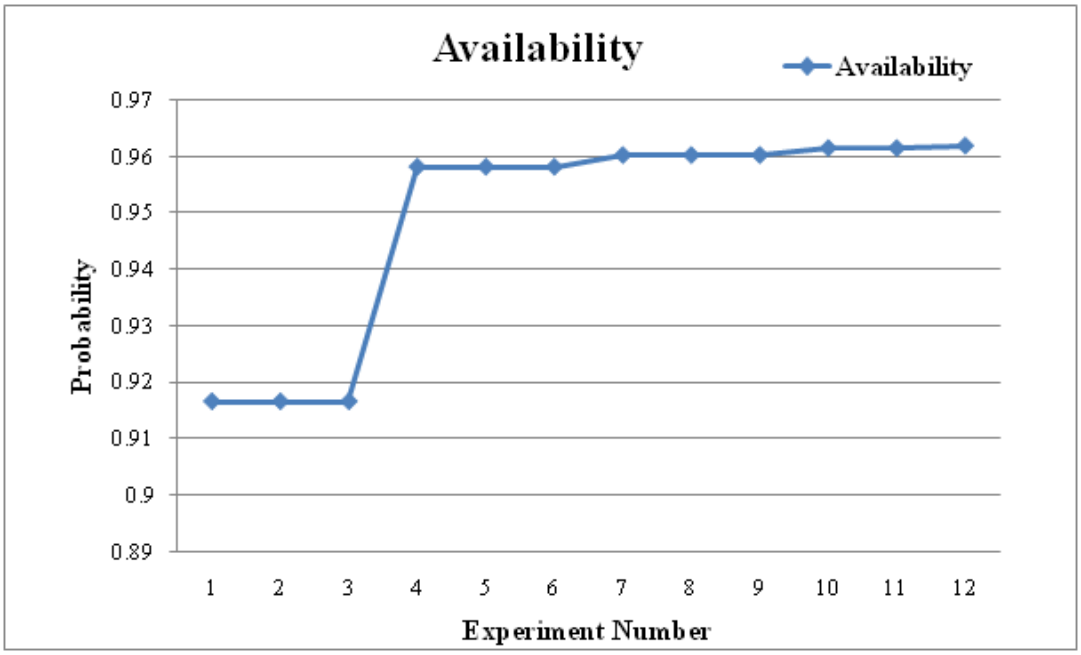

Figure 3.7 Comparison graph of Availability

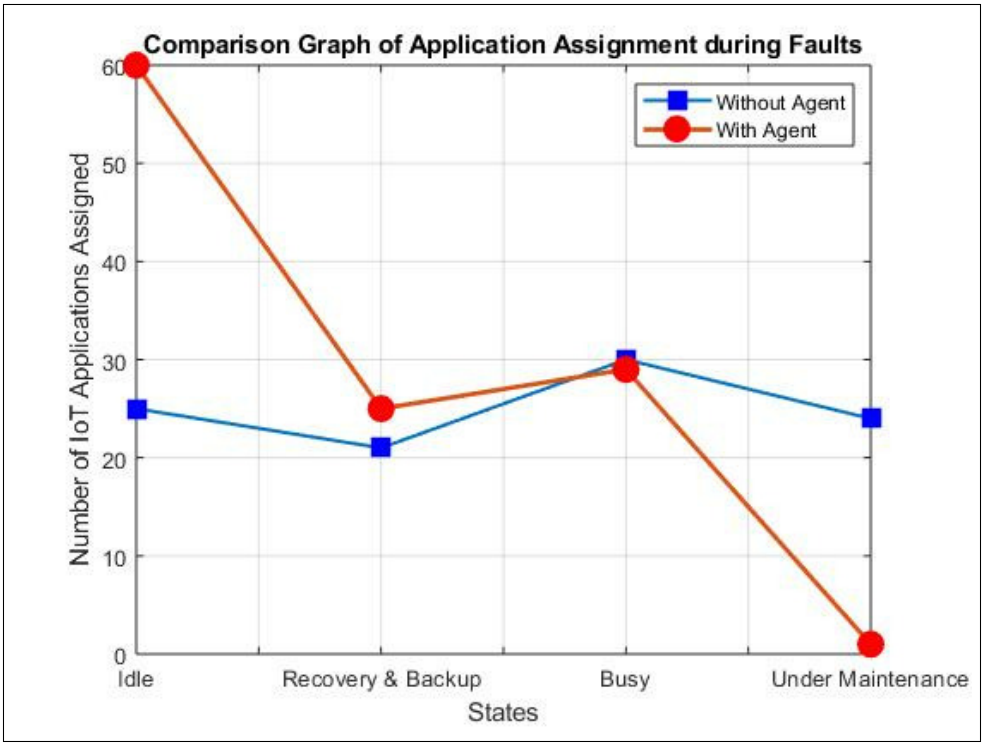

Figure 3.8 Comparison graph of application assignment during faults

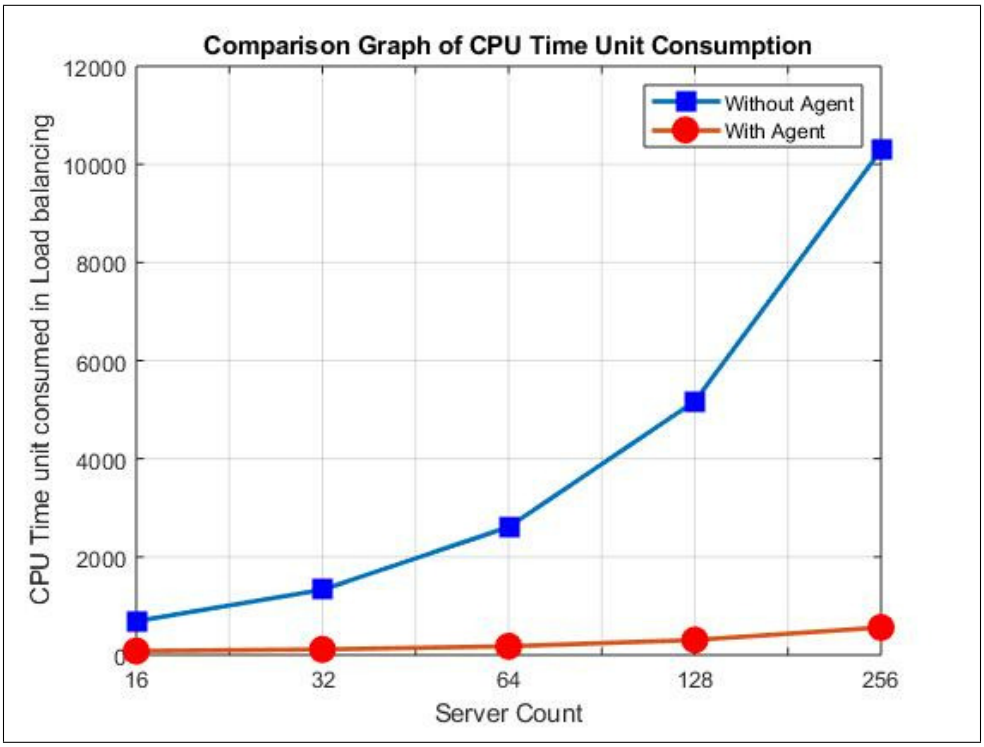

Figure 3.9 CPU time unit consumption

Chapter 4

Reliable Resource Allocation at Edge

As discussed in the earlier chapters, traditional IoT architectures connect the sensing devices to the Internet and send the generated data to a Cloud resource for processing. This methodology works well for the applications where the strict delay is not a concern. Cloud is not an ideal resource for the applications which require real-time responses. This is the reason, domains such as telecommunication, health care, real-time control etc. process the data closer to the origin [3]. This computational paradigm is termed as Edge Computing. After accepting Edge Computing, the next questions to be consider are

Are there sufficient Edge resources available to serve current and future IoT applications?

Can privately owned devices be used as the Edge resources in a reliable and efficient manner?

Gaber et.al. [37] proposed that the Edge Computing needs to leverage all the resources connected at the network edge such as laptop, tablets, desktops, smart-phones etc. Although, these devices are not always connected to the Internet, they can still be used opportunistically for various computational tasks. Nowadays, devices at the network edge such as laptops and desktop computers are being voluntarily offered by the owners for computational purposes. Efficient utilization of all these computational resources enhance the overall capacity of the edge system and allow it to scale. The traffic towards the core IP network can also be reduced significantly. In this regard, this chapter presents a self-organizing edge infrastructure laid on the principles of SDN. To realize it, an intelligent software-defined Edge Controller (EC) in the IoT environment is proposed, which configures the infrastructure to utilize all the available resources efficiently for data processing. This EC learns various parameters for resource optimization during initial runs. After acquiring sufficient information, an integer linear programming problem for minimization of time to complete requests is formulated and solved. The solution ensures fair allocation of the requests to various resources in such a way that the total time for request completion is minimized for all the nodes.

The *reliability* of the voluntary resources is a substantial issue which is covered in this chapter. Many times a computational resource accepts a request, but can not process it in a timely manner. It may happen because of the battery outage, scheduling of other higher priority tasks, connection re-establishment due to mobility etc. Assigning latency sensitive tasks to such devices is a risk. The formulation done for

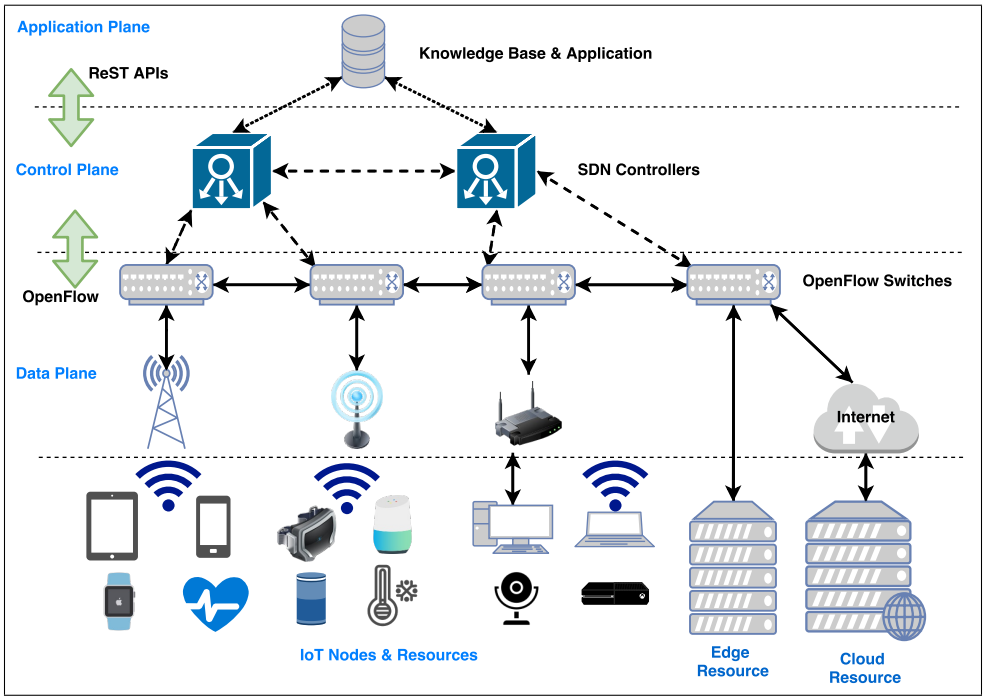

Figure 4.1 Software-defined IoT infrastructure.

minimization of time to complete requests has been extended to incorporate reliability measures on the EC. A multi-objective optimization problem is formulated and solved using Genetic Algorithm (GA). The solution to the problem provides the allocations which reliably forward the requests to the available resources.

Here on-wards two kinds of nodes have been considered in this chapter. The first kind of nodes, termed as IoT nodes, are the one which need to offload the computational tasks. These are sensing devices, which sense a physical phenomenon and transmit it to a server for further processing. In other words, IoT nodes are the clients looking for the servers. The second kind of nodes are termed as resources. These are the servers which provide the APIs for computation. Using APIs provided by the resources, IoT nodes request for various computation tasks.

4.1 Related Work

Resource allocation in edge and participatory computing has been studied by many researchers. In this section, research work by a few authors have been presented. Authors in [38] have presented a survey on emerging computing technologies such as cloudlet, fog computing and mobile Edge Computing. Importance of code mobility is discussed in [39]. The paper has emphasized to have a new mobile Cloud Computing infrastructure in the IoT environment. The positive impact of using user-controlled edge devices for the computation migration from the smartphone has been studied in [40]. Authors in

[41] have studied the computational offloading to the edge devices such as a router. Their proposed algorithm allows efficient resource utilization. Researchers in [42] have proposed the use of cloudlets on the edge of the network to assist IoT nodes in a better way compared to the clouds. Amin et.al. [12] have presented their work on the participatory Edge Computing for the local community services. In their work, they have achieved privacy of sensed data, data analytics, resilience and real-time response using participatory Edge Computing. Resource allocation and task offloading in mobile Edge Computing scenario is proposed by Tran et.al. [16]. A convex optimization technique is used for resource allocation and a heuristic algorithm has been proposed for task offloading.

The possibility of inclusion of SDN in Edge Computing for the mobile users is shown by Ahmet et.al. [43]. This review paper suggests the use of separate data and control planes to manage the movement of mobile IoT devices using Cloudlets. Authors in [44] have proposed an energy efficient routing protocol using the SDN controller. They have compared the proposed routing algorithm with the standard wireless sensor networks protocols such as Ad hoc On-Demand Distance Vector (AODV), Dynamic Source Routing (DSR) and Destination-Sequenced Distance-Vector Routing (DSDV) on various parameters such as throughput, end to end delay and packet delivery ratio. Qin et.al [18] have proposed a software-defined approach to manage heterogeneous IoT and sensor devices. This is done by providing the best matching resource for different classes of IoT devices.

Considering the aforesaid work, a novel software defined Edge Computing infrastructure has been presented, where the computational resources of the voluntary devices are utilized along with the dedicated edge and Cloud resources. This infrastructure allocates requests via learning different parameters in software-defined paradigm. The problem formulation tackles important issues such as minimization of request completion time and reliability of all the edge resources.

4.2 Proposed System for Resource Allocation with Request Completion Time Minimization

Figure 4.1 shows the proposed system where the IoT nodes and the resources are connected via OF switches at the data plane. OF switches are configured by the SDN controller at control plane using OF protocol. On the top of the control plane, application plane resides. It communicates with SDN controller using ReST APIs. Application plane gathers the data required for optimal resource allocation. Using the stored data, it finds out the optimal resource allocation and communicates the same to EC. Finally, EC pushes appropriate configurations to the OF switches. Figure 4.2 demonstrates the system flow diagram. This flow diagram covers all the steps involved in learning and resource allocation.

Here, the goal is to design an infrastructure, where the total time to process the requests made by IoT nodes can be minimized. In order to achieve this objective, the proposed system goes through a learning phase. Learning starts with gathering information about available resources and their capabilities. Using this information the EC configures the OF switches to forward all the requests made by IoT nodes to

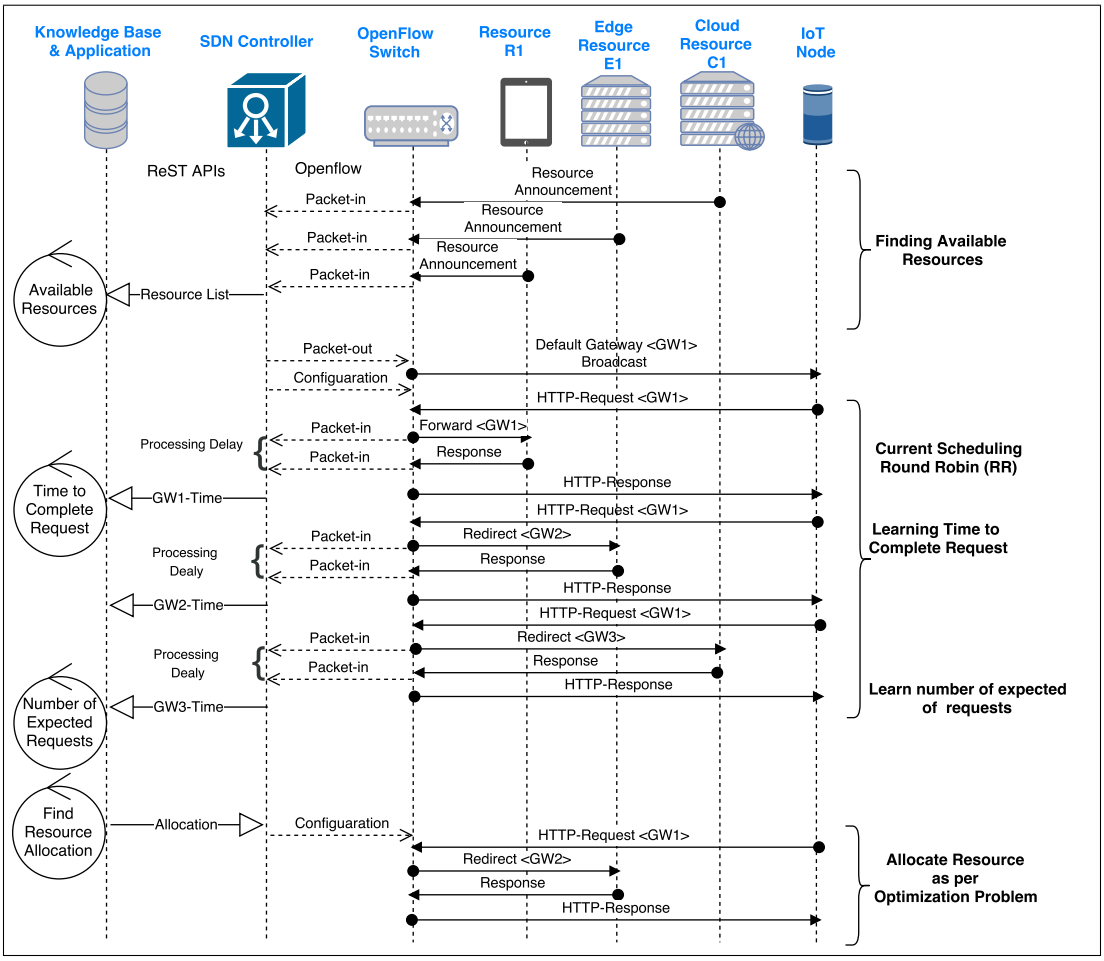

Figure 4.2 System flow summary.

all the resources in a round robin (RR) fashion. In this process, EC identifies the delay involved in completion of a request between an IoT node and a resource and expected number of requests from an IoT node. Once, the system has gathers all the required information, EC starts forwarding the requests in such a way that the total time to process the requests made by IoT nodes is minimized. This feature makes our controller self-organizing. The steps involved in the learning process have been described in detail, further in this section.

4.2.1 Finding Available Resources

The first step towards making the system self-organized is to gather complete information about the available resources in the system. Before scheduling the requests, the EC must be aware of the available resources for computation. In diverse IoT environment, different nodes may use different mechanisms and protocols for service advertisement and discovery. Nodes may use centralized or distributed approaches for resource discovery. In a Constrained ReSTful Environment (CoRE) [45] the devices use

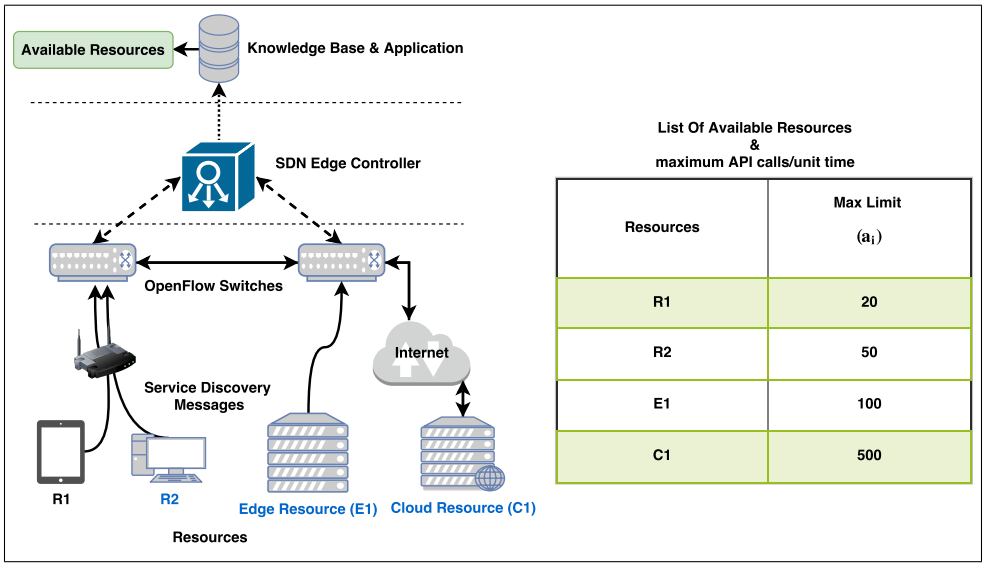

Figure 4.3 Learning resource availability and capability.

CoRE link format [1] for resource discovery. Nodes can also use the Resource Discovery Protocol (RDP) [46]. Both the protocols support the server/client model for resource discovery. In such centralized architecture, the RDP clients (or resources) send the resource information as a well-constructed query to the RDP server and the server stores this information in a resource directory. The nodes looking for resources send a query to RDP server and the server responds accordingly. A variety of other service discovery protocols in IoT setting can be found in [47, 48]. It is assumed that the RDP query will contain the maximum capability of the resource.

The information about service mechanisms and resource databases is provided as a configuration to the proposed system. At start-up, it parses the configuration file and configures the OF based SDN switches in the infrastructure to capture these resource information packets about the available resources. For example, say the devices announce their capability via RDP and this information is mentioned in the configuration. The switch flow tables are configured by the Edge Controller to receive an RDP announcement flows on the mentioned protocol and port. The Edge Controller parses these flows and updates its knowledge base about available resources. If there is an RDP server in the system, EC sends the RDP query to receive information about available resources via the packet-out mechanism. It is assumed that the entire system is also connected with the Cloud resources and the EC has this information. Figure 4.3 depicts the same idea.

4.2.2 Mapping Resources to IoT Nodes

Once EC has the list of available resources and the maximum requests they can handle, it starts forwarding requests coming from IoT nodes to the available resources in a RR manner. Before doing

[1] https://tools.ietf.org/html/draft-ietf-core-resource-directory-09

this, EC configures the SDN switches to announce the default resource address for all the available services. IoT nodes looking for resources, receive these notifications and start sending requests to announced default gateway. The EC configures the SDN switches in such a way that the requests go to all the available service providers one by one. For redirecting the requests to other resources, switches change the destination address of a request from default gateway to a selected resource.

For example, a camera-equipped IoT node wants the captured images to be classified. It receives the default resource information and starts sending requests to the mentioned address. Due to the switch configuration, the destination of the request changes and the first request is forwarded to first available image classification service provider. The second request is forwarded to another service provider. This process goes on for subsequent requests in a RR fashion. With the analysis of requests and responses EC maps the IoT nodes to the resources with various metrics such as processing delay and the number of expected requests from an IoT node.

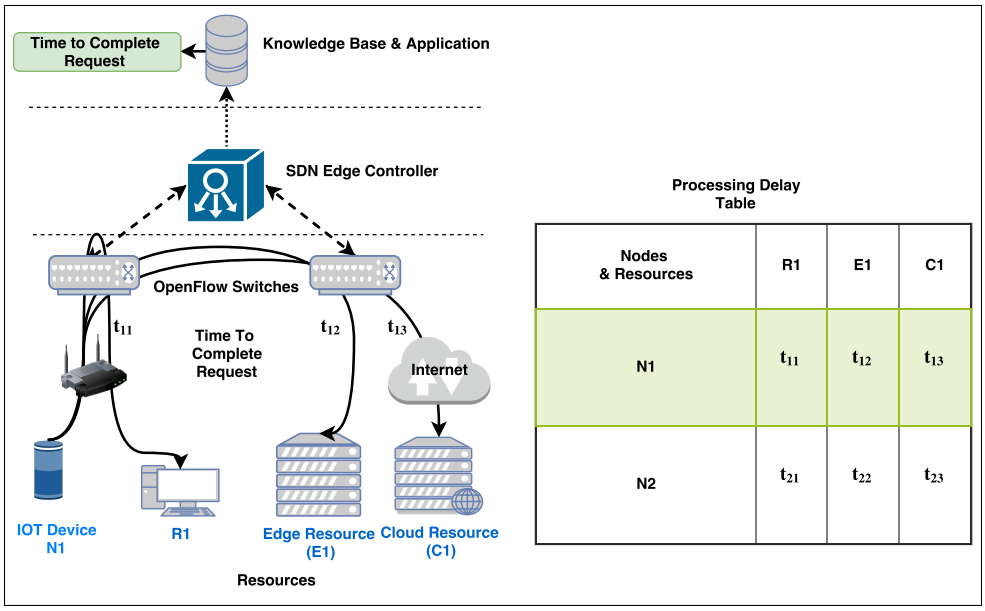

Figure 4.4 Learning request completion time (MTTC) from IoT node to the resources.

4.2.2.1 Learning Request Completion Time

As discussed earlier that the IoT environment is heterogeneous. Service providing node can be connected via different media access protocols. They can have different processing capabilities. They can be static or mobile. All these situations change the time for processing a request. Via forwarding requests to all the resources in a RR manner, our controller learns about the expected time taken for completing a request. Figure 4.4 explains the idea pictorially. In order to find the lapsed time in processing a request, EC configures the switch to send notification packets to itself as soon as a request is forwarded to a resource and resource sends the response back. Looking into the time difference, EC identifies the

delay involved. This delay is also termed as Mean Time to Complete (MTTC). This process is repeated for every pair of IoT node and resource.

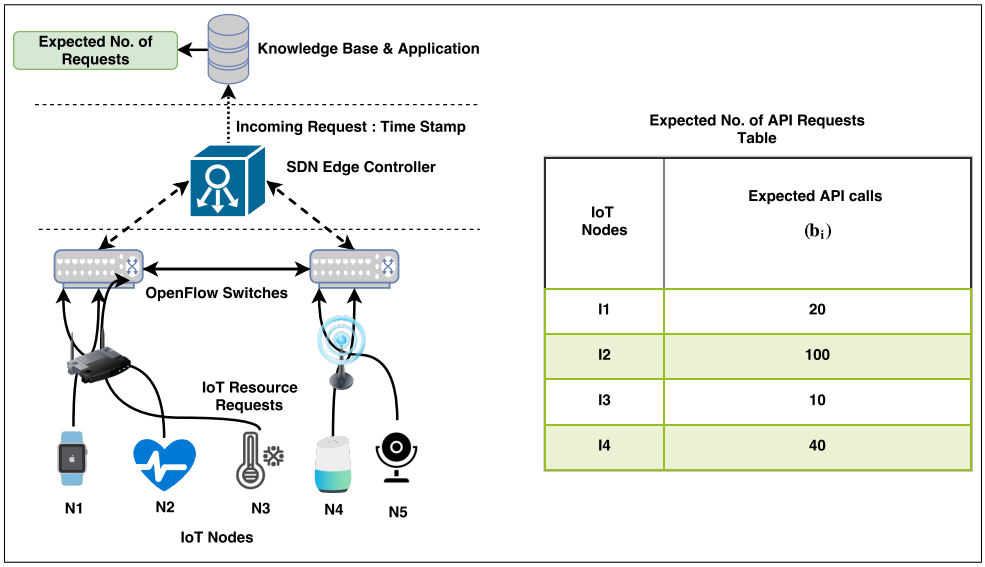

Figure 4.5 Predicting number of expected requests from IoT nodes.

4.2.2.2 Predicting the Number of Expected Requests

In the process of forwarding requests to all the available resources, the EC also determines the number of expected requests from an IoT node in a duration. While forwarding requests in a RR manner, the EC receives the request notification (as EC configured the switches for learning the MTTC, mentioned in previous subsection 4.2.2.1). The EC saves these records with a time stamp. When sufficient data is collected, the application plane analyzes the data and forecasts the expected number of requests in future. Neural networks or deep learning based time series analysis techniques can be employed by the application plane to predict the expected number of requests in the future. Figure 4.5 explains the same.

4.2.3 Resource allocation as an optimization problem

After going through the learning phase, the system acquires the information about the maximum capacity of m service providing nodes represented as a vector $[a_1, a_2, \ldots, a_m]$. It also gains knowledge about number of expected requests from n IoT nodes, depicted as $[b_1, b_2, \ldots, b_n]$. It maintains a matrix $[t_{ij}]$ $\forall\ i = 1, 2, \ldots, m$ & $j = 1, 2, \ldots, n$ representing average time taken for completing requests of IoT nodes to service proving nodes (refer figure 3.6). x_{ij} is the number of requests forwarded to a resource i from an IoT node j. As resource scheduling techniques based on mean time to complete are more effective than RR or random selection techniques [49], EC switches the allocation method from the RR to the MTTC (or request completion time minimization) technique via the following formulation

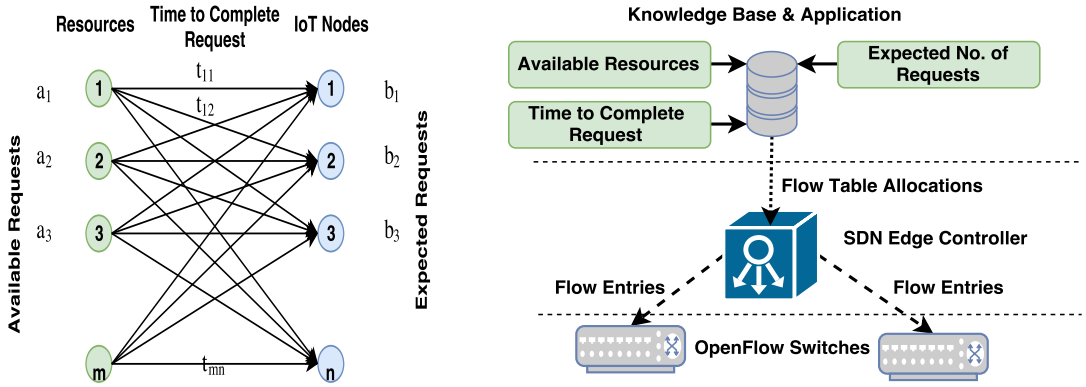

Figure 4.6 The resource allocation problem and the system with acquired knowledge.

$$\text{minimize} \sum_{j=1}^{n} t_{ij} x_{ij} \; 8 \, 1 \quad i \quad m \tag{4.1}$$

subject to constraints

$$\sum_{j=1}^{n} x_{ij} \quad a_i \; 8 \; i \; = \; 1, 2, \ldots, m \tag{4.2}$$

$$\sum_{i=1}^{m} x_{ij} \; = \; b_i \; 8 \; j \; = \; 1, 2, \ldots, n \tag{4.3}$$

$$x_{ij} \quad 0 \; 8 \; i \; = \; 1, 2, \ldots, m \; \& \; j \; = \; 1, 2, \ldots, n \tag{4.4}$$

Equation (4.1) is the objective function. It tries to minimize the total time taken for processing the requests for all the available IoT nodes. Equation (4.2) ensures that the requests to a service providing node is not more than its maximum capacity. Similarly, equation (4.3) divides the load from an IoT node to service proving nodes. Equation (4.4) puts a lower bound on the allocations. Clearly, to have a feasible solution following the condition must be satisfied

$$\sum_{i=1}^{m} a_i >= \sum_{j=1}^{n} b_j \tag{4.5}$$

If the above condition is not satisfied, more service providing nodes are required. The EC can request neighbouring edge servers for more serving units or it can reserve the Cloud resources to accommodate requests. This shows the proactive behaviour of the proposed system. The solution to the above optimization problem provides flow allocation from IoT nodes to resources. This allocation scheme runs at user-defined intervals and the application plane pushes new allocations to EC. EC changes switch configurations accordingly.

4.2.4 Results

The analysis of the proposed system has been done by simulated it using MATLAB and mininet [50]. First, the formulated linear programming problem has been solved in MATLAB. Results are verified for request completion time minimization and fairness. Later, a scenario is created in mininet and request completion time minimization criterion is tested.

4.2.4.1 MATLAB results and analysis

The assignment problem has been solved using MATLAB function intlinprog [51] with different values of requests and processing delay (or MTTC values). The function uses a dual-simplex algorithm [52] for solving the problem.

Table 4.1 MATLAB simulation parameters.

Case	Requests from IoT nodes	Max Capacity of Resources	MTTC(ms)
a	200-300	1000-2000	7-9
b	200-300	1000-2000	7-17
c	1000-2000	2000-4000	7-9
d	1000-2000	2000-4000	7-17
e	10-300	1000-2000	7-9
f	280-300	1000-2000	7-9

The information about the number of expected requests from IoT nodes, resource capabilities and the MTTC values are randomly generated as per table 4.1. The table has six test cases. These cases verify the system behaviour in different conditions such as low and high expected requests from the IoT nodes, the low and the high difference in the MTTC values, and the low and the high difference in the requests from the IoT nodes. The values mentioned in the tables are treated as knowledge gained by the system over time. The program uses these values and computes the allocation. The test cases are repeated for the number of IoT nodes (50, 100, 250, 300 and 500).

From results in figure 4.7, it is clear that the average response time in case of the MTTC is very low as compare to RR. Until Edge Controller does not have a complete knowledge base, it uses the RR technique to serve the IoT nodes. As soon as it acquires the information, flows are redirected using the MTTC technique which reduces the time taken for request completion significantly. From plots, it is clear that the MTTC method provides an output at a constant rate, independent of the difference in the MTTC values. IoT nodes face equal delays for request processing in high as well as low difference MTTC values. The simulations are run with a combination of less/more number of requests and low/high variance in the MTTC values.

For testing fairness, cases (a) and (b) from table 4.1 have been taken. In these cases all the IoT nodes generate an equal number of requests (300). All the nodes in the RR method take different times for completion of their requests, while the MTTC method takes an almost an equal amount of time. When the difference in the MTTC values are low, the MTTC scheduling allocates resources in such a way

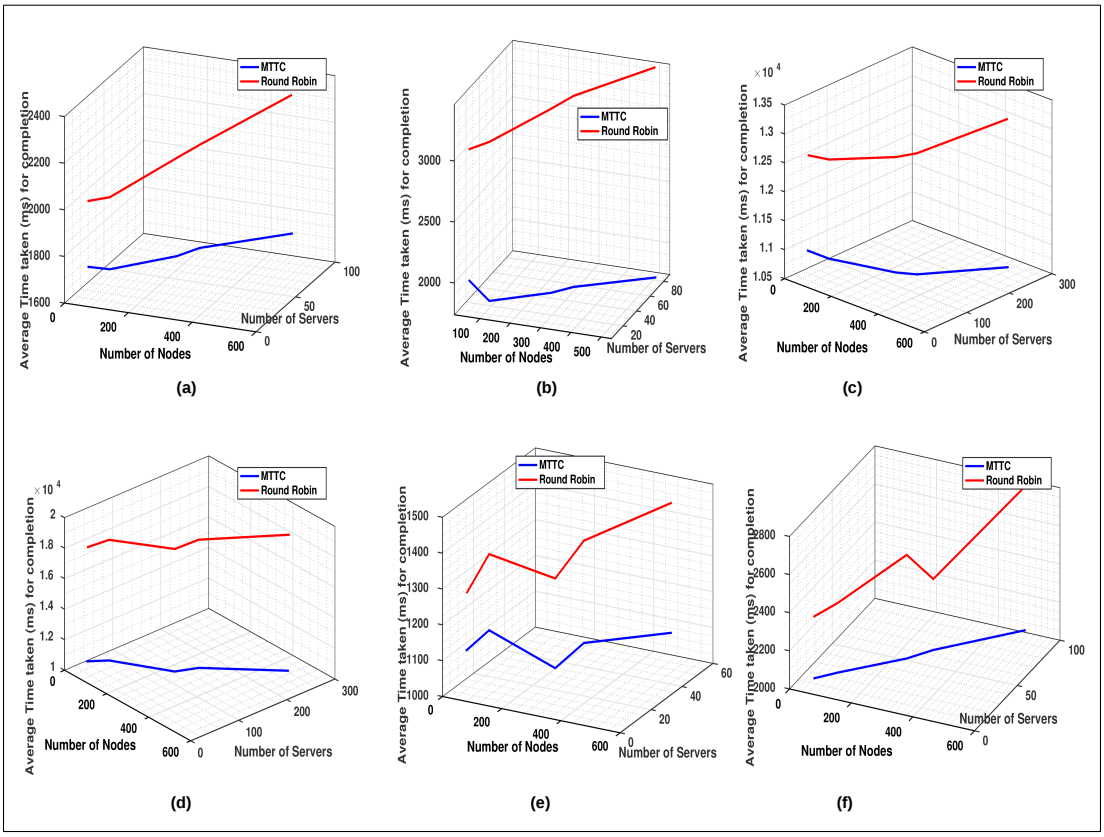

Figure 4.7 Comparison between round robin and MTTC methods. a) less no. of requests and less MTTC difference, b) less no. of req. and high MTTC diff., c) more no. of req. and low MTTC diff., d) more no. of req. and high MTTC diff., e) high req. diff and low MTTC diff., f) low req. diff. and low MTTC diff.

that all the IoT nodes experience an equal amount of times for their request completion. Even in case of high variation in the MTTC values, the majority of the IoT nodes have an equal amount of request completion delays with the MTTC scheduling. In the RR allocation, the IoT nodes have different delays for both the cases. Figure 4.8 confirms this behaviour.

4.2.4.2 Mininet Analysis

Mininet is a network emulator in which a network can be created with virtual hosts, switches, controllers and links. The switches in mininet support OpenFlow. For our scenario, mininet is configured with four IoT nodes (h1-h4), three resources (r1-r3) and one SDN controller as shown in figure 4.9. The link delays for resources are configured to have different values. In the setup, r1, r2 and r3 are configured to have 50ms, 200ms and 500ms of link delays respectively. It helps in creating different response times for different resources. A concurrent hypertext transfer protocol (HTTP) server is run on all the available resources. IoT nodes run an HTTP client, which periodically sends requests to the resources as per the table 4.2.

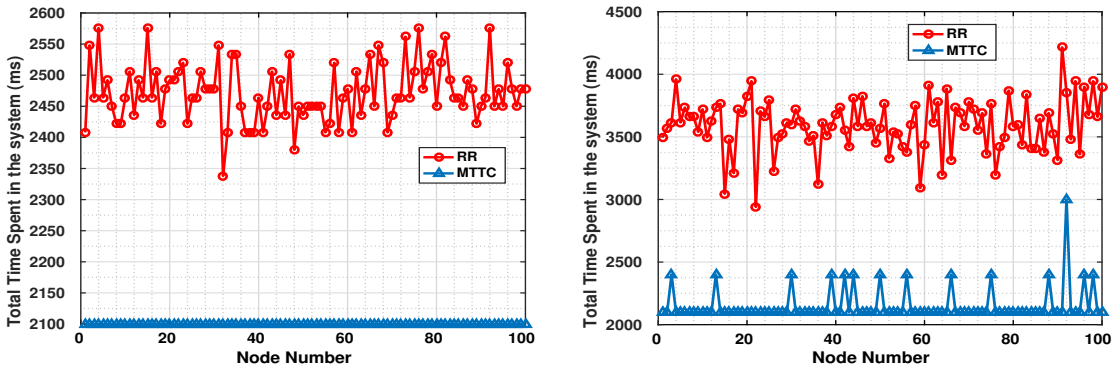

Figure 4.8 The fairness comparison. MTTC values in the left figure vary in rangev(7 – 9) and in range (7 – 17) in the right figure.

Table 4.2 Mininet simulation parameters

Node	Link Delay (ms)	Request or Response (/scenario)
h1	100	14,15,30
h2	100	12,15,2
h3	100	20,15,8
h4	100	10,15,15
r1	50	20,20,20
r2	200	20,20,20
r3	500	20,20,20

First, the system looks for available resources. It captures the resource advertisement packets and knows about the maximum capacity of available resources. With this information, the SDN controller configures the switch with RR mechanism and learns other required parameters. For every forwarded request it waits for the response and measures the MTTC values. In the simulation IoT nodes have configured to send requests in a fixed interval. For simplicity, this information is communicated to EC. Table 4.2 shows all the simulation parameters. Request or Response per scenario column tells the number of requests sent by an IoT node (h1-h4) and maximum requests resources (r1-r3) can entertain in three scenarios. Other columns are self-explanatory.

Once it has all the values, the data is fed to MATLAB to obtain allocations as per MTTC method. The allocations received from the MATLAB are given to the controller and the controller configures the switches to allocate resources accordingly. Figure 4.10 shows that the time taken for request completion by the MTTC method is very less than RR method. It clearly shows that the proposed system minimizes the delay.

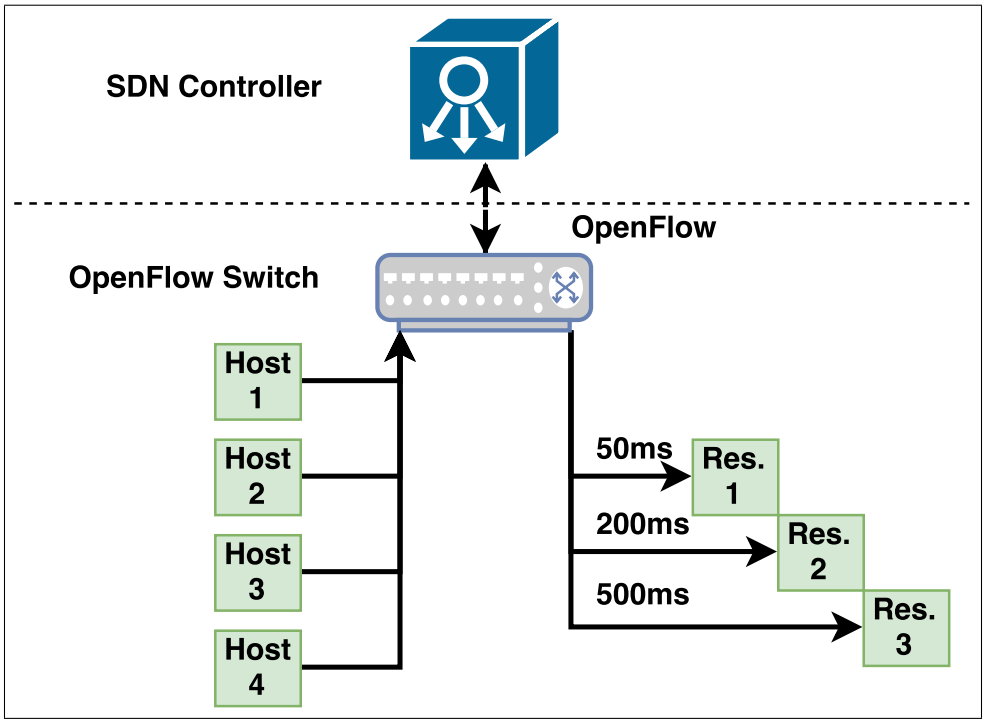

Figure 4.9 Mininet configuration. Hosts are generating requests and resources are responding.

4.3 Proposed System for Resource Allocation with Reliability

In the previous section, the resources were allocated in such a way that the allocation is fair and the total delay involved in processing is minimized. However, the formulation does not consider the reliability of the edge resources. Majority of the resources at the network edge, which are provided voluntarily, are unreliable. Highly mobile resources are prone to drop the connection. These resources may refuse to entertain requests due to the scheduling of high priority local tasks, connection unavailability or low battery conditions. Hence, there is a need to update the system to incorporate resource reliability information. In this section, the previously mentioned system is updated to forward the requests in such a way that the total delay involved is minimized along with the minimization of allocated requests to the resources with high probability of failure.

4.3.1 Knowing Resource Reliability

Multiple factors can decide the reliability of a resource. Mobility, power (battery), workload, security etc. are few such parameters. The reliability of a resource goes down in case of low battery condition. Similarly, if it is switching across access points or a user is running computational extensive application at the moment, it is risky to forward a request to that resource. In our study, the reliability of a resource is termed as the number of responses sent in a timely manner for each request. As mentioned in the above section, the EC configures the switch to find out the processing delay involved in forwarding a

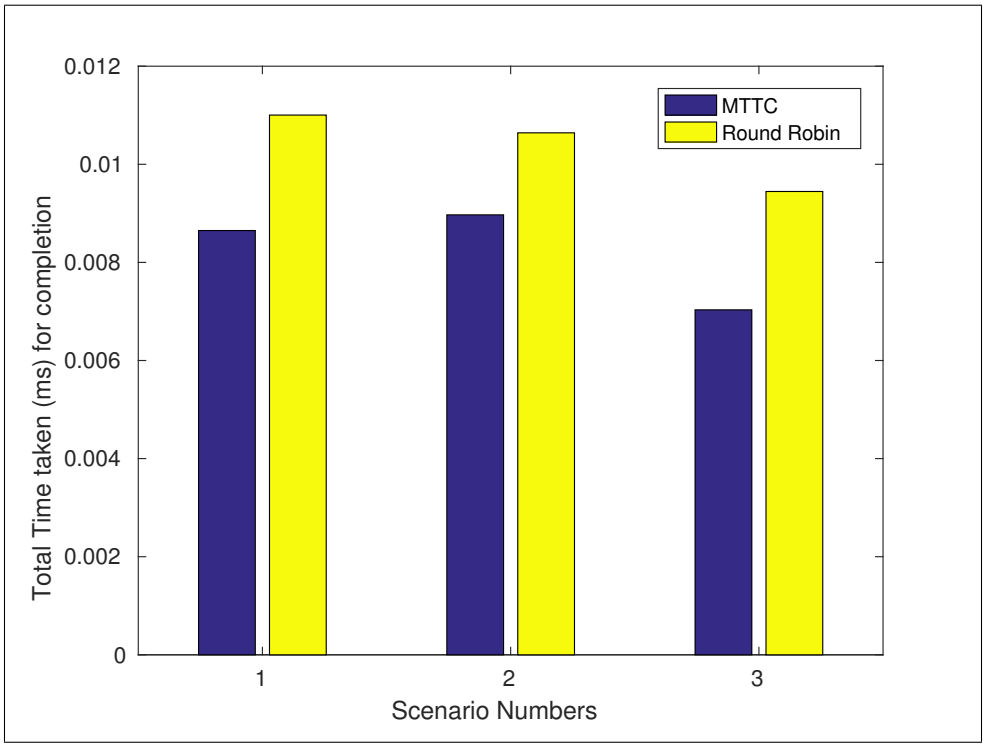

Figure 4.10 Time saving in three different scenarios.

request from IoT node to a resource. EC uses this timing in realizing the reliability of a resource. If the time difference between a request and response is more than a threshold, it is very unlikely to meet the application requirements. Hence, the EC reduces the reliability of such a resource or in other words it increases the probability of failure of that particular resource. Actual reliability value is calculated as the ratio of timely sent responses (n_{res}) to the total sent requests (n_{req}) i.e. n_{res}/n_{req} (refer algorithm 4.3.1). The same idea is depicted in figure 4.11.

Algorithm 2 Pseudo-code for calculation of resource reliability.

for *each IoT request* **do**
  Request Count = n_{req} + + Timestamp of incoming request = t_1 Timestamp of outgoing request = t_2 Processing Delay = t_2 t_1 **if** *Processing Delay* <= *Min Threshold* **then**
    Response Count = n_{res} + +
  else
    /*No change in Response Count */
  Reliability = n_{res}/n_{req}

4.3.2 Resource allocation as multi-objective optimization problem

The application plane formulates a dual objective resource allocation optimization problem as per figure 3.12. For m nodes it has probability of failure represented as $[f_1, f_2, \ldots, f_m]$. If x_{ij} be the number

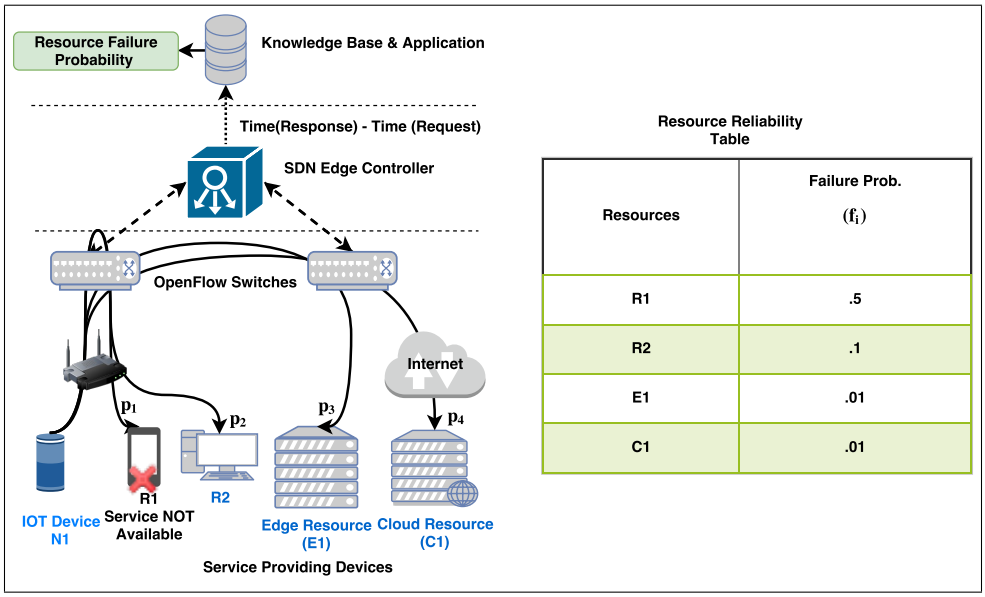

Figure 4.11 Predicting resource reliability.

of allocated requests to a service proving node i from an IoT node j, multi-objective optimization problem is formulated as following

$$\text{minimize} \sum_{j=1}^{n} t_{ij} x_{ij} \; 8\, 1 \quad i \quad m$$

$$\text{and} \tag{4.6}$$

$$\text{minimize } \Pi_{i=1}^{m} f_i^{\sum_{j=1}^{n} x_{ij}}$$

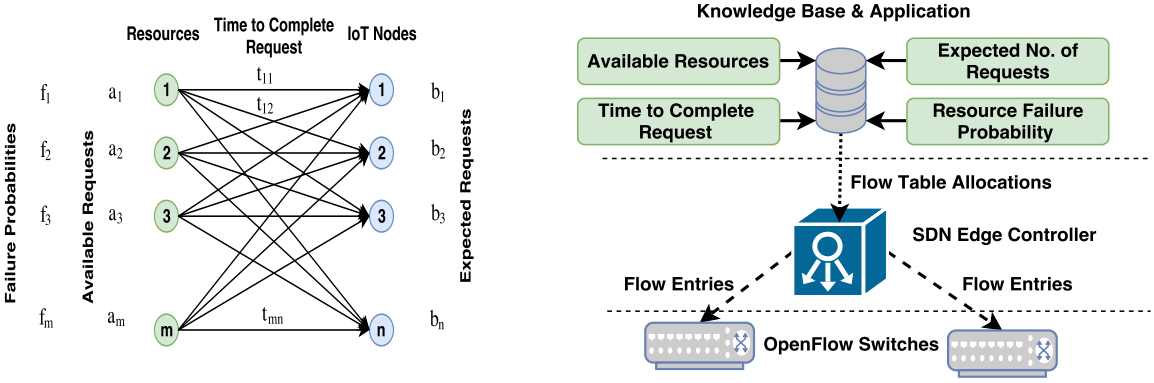

Figure 4.12 Resource allocation problem and system with acquired knowledge with reliability information.

subject to constraints

$$\sum_{j=1}^{n} x_{ij} \quad a_i \; 8 \; i = 1, 2, \ldots, m \tag{4.7}$$

$$\sum_{i=1}^{m} x_{ij} = b_i \; 8 \; j = 1, 2, \ldots, n \tag{4.8}$$

$$x_{ij} \quad 0 \; 8 \; i = 1, 2, \ldots, m \; \& \; j = 1, 2, \ldots, n \tag{4.9}$$

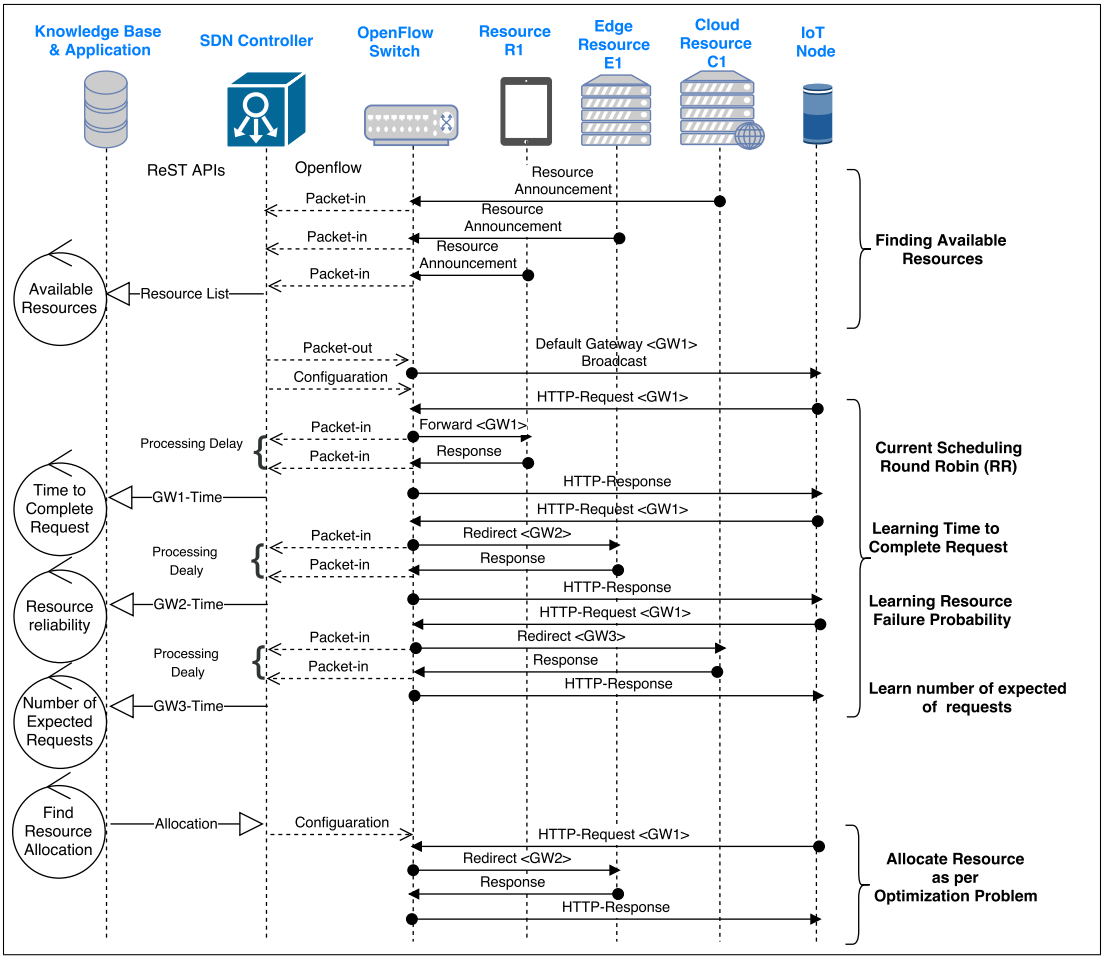

Figure 4.13 System flow summary after resource reliability information.

Equation (4.6) is the objective function. It tries to simultaneously minimize the total time taken for processing the requests for all the nodes and the failure probability associated with that node. Constraints are same as before. With the inclusion of resource reliability in learning steps, the updated system flow diagram looks like figure 4.13.

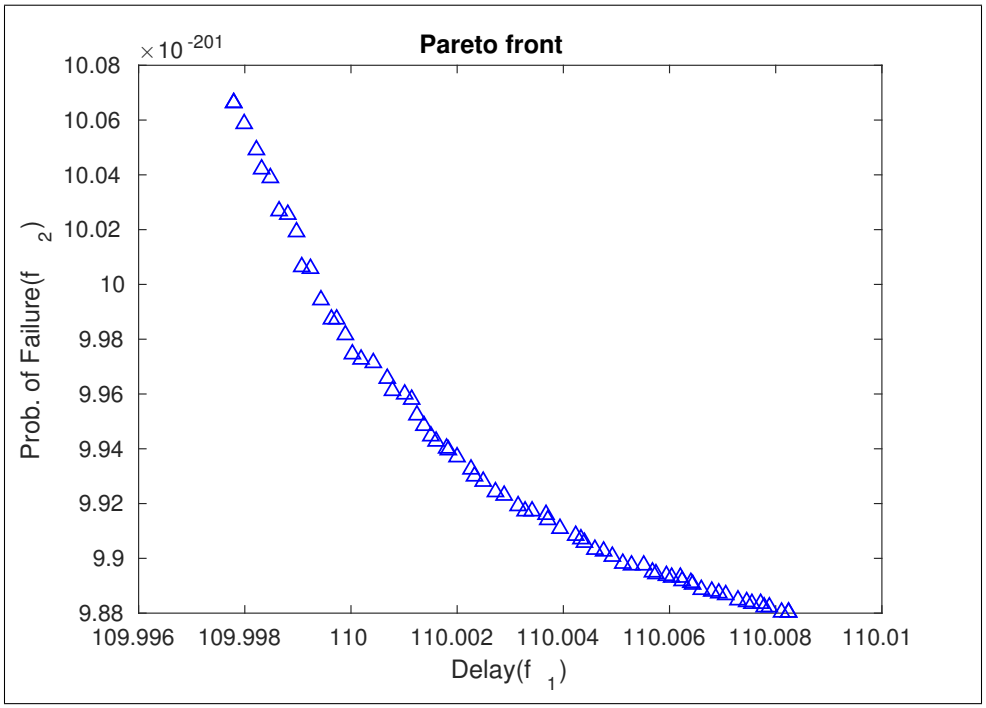

Figure 4.14 Pareto front of a case.

4.3.3 Results

For solving the optimization problem formulated in section 4.2.3, Genetic Algorithm [53] has been used. The problem has been coded and solved in MATLAB using function gamultiobj() [54]. The initial configuration parameters can be found in table 4.3. The solution to the above optimization problem gives multiple Pareto optimal solutions in terms of delay and probability of failure. The Pareto front can be observed in figure 4.14. To narrow down our search to achieve the appropriate optimal values of allocation, k-means clustering has been utilized. In this approach, the solutions are kept in three clusters. The First cluster has values with low delay and high probability of failure. Contrary to that, the third cluster has values with high delay and low probability of failure. Unlike first and third clusters, the solutions in the second cluster are not extreme. Hence, a solution from cluster-2 is picked randomly and the requests from the IoT node to the resources are allocated accordingly.

To verify the proposed allocation strategy, the scenario as per figure 4.15, has been kept under study. N1, N2 and N3 are three data generating IoT nodes. Processing is required on the generated data. For entertaining the requests coming from IoT nodes, resources R1, R2 and R3 are available in the infrastructure. R1 is a dedicated edge resource and it can respond to 100 requests per unit time. It is immobile and has connectivity via Ethernet. It has redundant power and storage. Hence, the probability of failure is very low (.01). R3 is a cloud-connected resource. It is highly scalable and can process 500 requests per unit time. The probability of R3's failure is also very low (.01). Resource R2 is a mobile node. It is battery powered, connected via WLAN or 4G link. When R2 is mobile, it switches across

Table 4.3 Initial parameters used in GA for a multi-objective optimization problem.

Parameter	Value	Details
CreationFcn	@gacreationuniform	Initial population is chosen with uniform distribution.
CrossoverFcn	@crossoverintermediate	Children are created using weighted average of parents.
CrossoverFraction	0.8	Next generation population fraction.
DistanceMeasureFcn	@distancecrowding,'phenotype'	Measure the distance of individuals in function space.
FunctionTolerance	10^{-4}	If relative change is less than this value, the algorithm stops.
MaxGenerations	1800	Maximum iterations.
MutationFcn	@mutationadaptfeasible	Random changes in individuals depend upon last successful/unsuccessful generation.
PopulationSize	200	Initial population size.
SelectionFcn	@selectiontournament	Parents for the next generation are chosen by a tournament of size 4.

access multiple points. Hence, the reliability of R3 is lesser than R1 and R3. The probability of failure for R2 is 0.1 and it can process 20 requests per unit time. R1 is installed close to N1 and N2. Therefore, N1 and N2 receive quick responses for the requests sent to R1. If the requests are sent to R2 or R3, the time to receive the response increases. Similarly, N3 is closer to R2, so the delay involved in receiving the responses for R1 & R3 is higher as compared to R2.

The system starts usually by collecting the available resource information. With available resource information in hand, the EC configures the switches to forward the requests in a RR manner. In the learning process the system not only learns about the mean time to complete requests and the expected number of requests from each IoT node but also, it keeps track of the probability of failure of a resource using the algorithm 4.3.1. With this information, the application plane solves the multi-objective optimization problem using GA and forwards the allocation to the EC. The EC receives the optimal allocations from the application plane and configures the OF switches accordingly. For testing the formulation following two conditions have been considered.

4.3.3.1 Varying probability of failure

In this case, the allocations are observed with a varying probability of failure of resource R2. As discussed previously, R2 is a mobile resource and it may not be able to process allocated requests in a timely manner due to conditions such as battery outage, change of access point etc. Hence, it is necessary to understand the resource allocation changes with a varying probability of failure of resource R2. The number of expected requests from N1, N2 and N3 are 50, 100, and 30 respectively. The R2's

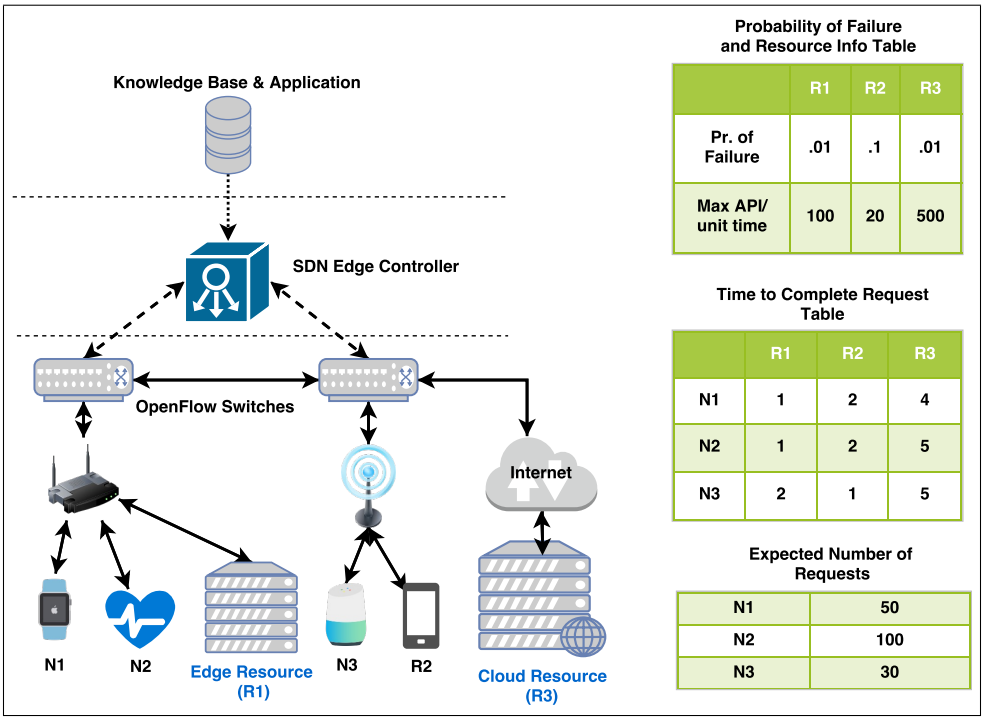

Figure 4.15 Scenario under test.

probability of failure is varied from 0.1 to 0.9. Figure 4.17 shows the various allocations with increasing probability of failure of R2.

Intuitively, with the rise in the probability of failure of R2, the EC should forward less number of requests to R2. Results in figure 4.17 confirm this behaviour. Further, EC transfers the allocations from R2 to R3. R1 is fully utilized, so there are no request shifts from R2 to R1. Figure 4.16 depicts the number of allocated requests to R2 and R3 with an increase in R2's probability of failure. It is clearly seen that the fall in the number of allocated requests to R2 is equal to allocations added to R3.

4.3.3.2 Varying Number of Expected Requests

As the expected number of requests from various IoT resources may vary over time, application plane updates EC about optimal allocations for a given time duration. With this information, EC updates the switch configurations and the SDN switches forward the requests accordingly. In this scenario, the value of the probability of failure and delay for request completion do not change. Only the expected number of requests from N1, N2 and N3 vary in given time duration. Figure 4.19 shows the allocated requests from IoT nodes to resources. These values have been obtained from initial knowledge represented in figure 4.15.

These results validate our formulation. As R1 is the best available resource, it is allocated the maximum number of requests. When total demand is less than or equal to R1's capacity, all the requests are assigned to R1 only. Case c and f in figure 4.19 confirm this behaviour. As demand increases and it

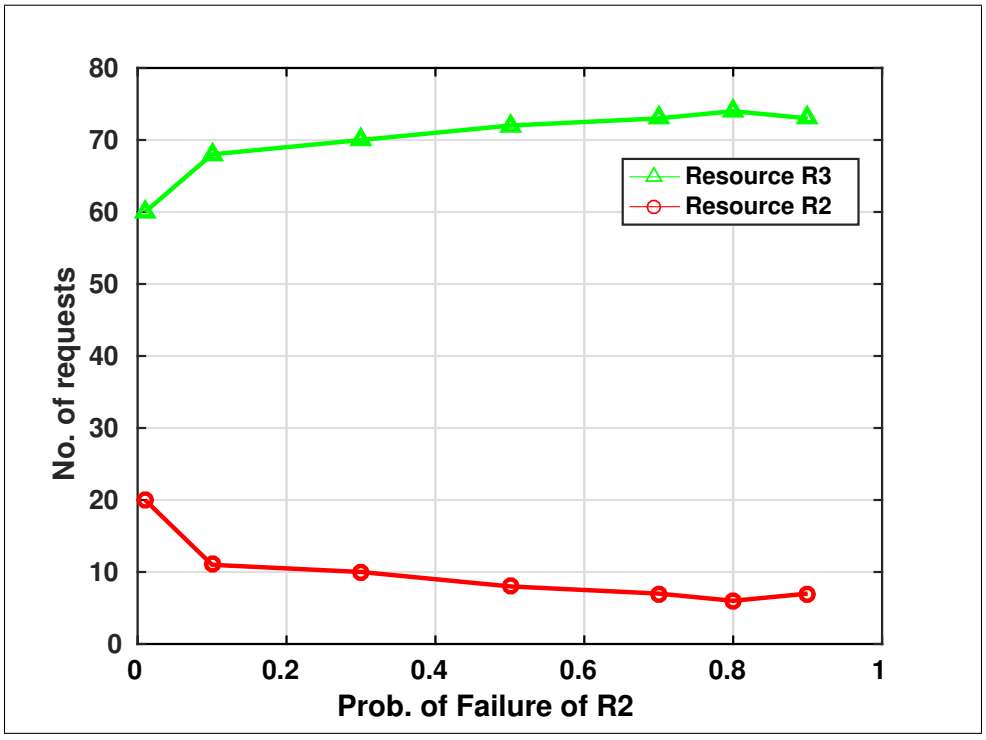

Figure 4.16 Probability of failure of R2 and the number of allocated requests.

reaches beyond R1's capacity, requests are forwarded to R2 and R3. Hence the best available resource is highly utilized. Figure 4.18 shows resource utilization for the same scenario.

	R1	R2	R3
N1	24	2	24
N2	60	9	31
N3	17	0	13

(a)

	R1	R2	R3
N1	25	1	24
N2	60	8	32
N3	15	1	14

(b)

	R1	R2	R3
N1	22	2	26
N2	64	6	30
N3	14	0	16

(c)

	R1	R2	R3
N1	24	2	24
N2	62	5	34
N3	15	0	15

(d)

	R1	R2	R3
N1	23	2	24
N2	62	4	34
N3	14	0	15

(e)

	R1	R2	R3
N1	22	3	25
N2	64	4	32
N3	14	0	15

(f)

Figure 4.17 Number of requests allocated to an IoT node to resource with the varying reliability of R2. The probability of failure for resources R1, R2 & R3 are (a) [.01 .1 .01] (b) [.01 .3 .01] (c) [.01 .5 .01] (d) [.01 .7 .01] (e) [.01 .8 .01] (f) [.01 .9 .01]

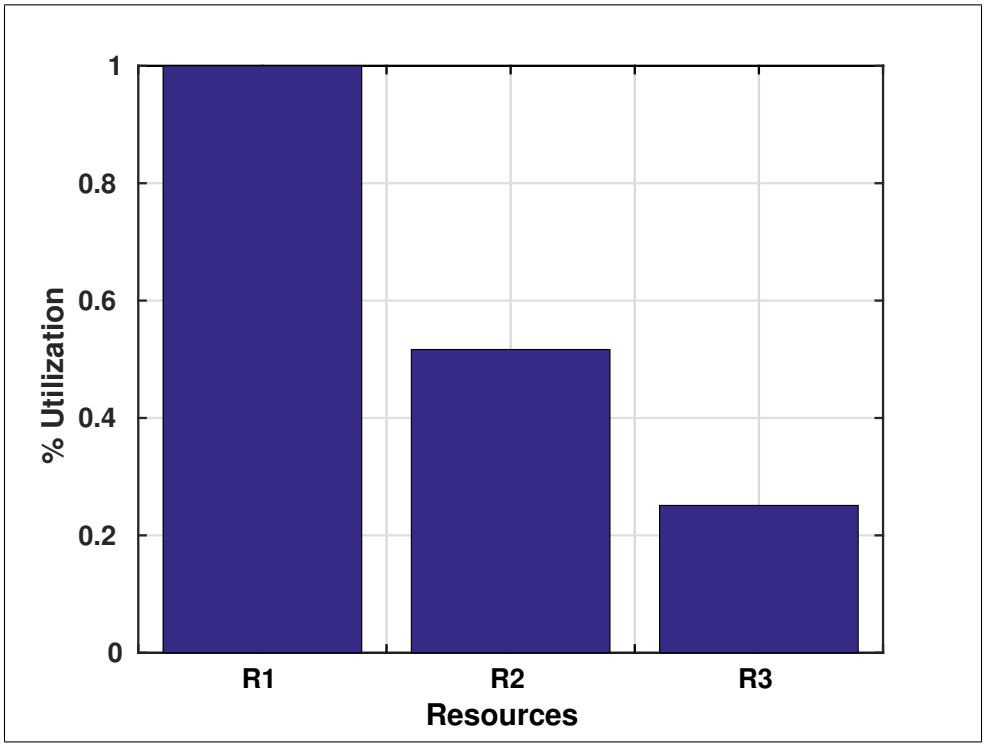

Figure 4.18 Overall resource utilization in case of variation in the number of expected requests.

	R1	R2	R3
N1	21	6	22
N2	66	9	25
N3	13	5	12

(a)

	R1	R2	R3
N1	6	5	9
N2	71	0	9
N3	18	7	16

(b)

	R1	R2	R3
N1	10	0	0
N2	80	0	0
N3	10	0	0

(c)

	R1	R2	R3
N1	46	0	54
N2	45	1	54
N3	9	19	72

(d)

	R1	R2	R3
N1	46	0	154
N2	45	0	155
N3	9	20	171

(e)

	R1	R2	R3
N1	0	0	0
N2	0	0	0
N3	100	0	0

(f)

Figure 4.19 Number of requests allocated to an IoT node to resource for varying expected number of requests. The requests made from N1, N2 & N3 for respective cases are (a) [50 100 30] (b) [20 80 40] (c) [10 80 10] (d) [100 100 100] (e) [200 200 200] (f) [0 0 100]

Chapter 5

Reliable Landslide Early Warning System - An Application

In many parts of the world including south-east Asia, a lot of landslides occur every year. They cause heavy loss to human lives and infrastructure. There are a few IoT technologies exist that allow landslides' monitoring to save lives and money but there is a requirement of a *reliable* Landslide Early Warning System (LEWS). The system must be reliable in terms of accurate alarm generation and must not fail in any given situation. This chapter is focused on the development of a *Reliable LEWS*. For achieving reliability, Machine Learning (ML)[1] is used to analyze the sensed landslide data and to generate reliable alarms. Along with the reliable alarms, the system must be reliable in terms of its availability/up-time. The communication between the IoT nodes and the Cloud is vulnerable to link failure due to various types of disruptions in mountainous regions. In real life scenarios, this connection loss might hinder the decision making at the Cloud due to the non-reception of needed data at the right time. This decreases the *reliability* of LEWS. With reliability, the system must be *fault-tolerant* as well. This chapter demonstrate the implementation of reliable data processing so that even if the connection is lost between the Source/Coordinator Node and the Cloud Server, the data can still be processed and feedback are obtained. For this, Edge Computing has been proposed as a solution to complement Cloud Computing. During implementation, the Edge Server has a limited computing and storage resources, but enough to process and analyze landslide data such as rain-fall, pore pressure, moisture, and displacement, to produce meaningful results similar to the Cloud. The system keeps on working even if there is a network failure between Edge Server and Cloud Server or Cloud Server crashes. Furthermore, it improves the availability of processing capabilities by switching to an alternate Edge Server in case of an Edge failure at a landslide location to make the system fault-tolerant. This system is a considerable improvement over any existing Landslide Warning Systems. It is very helpful in reliably and efficiently detecting landslides with a mechanism to deal with faults.

[1] As it is out of the scope of this thesis, this chapter does not focus on the ML part of the implemented system.

5.1 Background

5.1.1 Overview of Landslide

Landslides are one of the most dangerous and destructive natural hazards that cause significant damage to economic objects and human lives. The movement rate of landslides varies from the slow movement of material in millimetres/centimetres range per year to a sudden avalanche of a large quantity of debris [55].

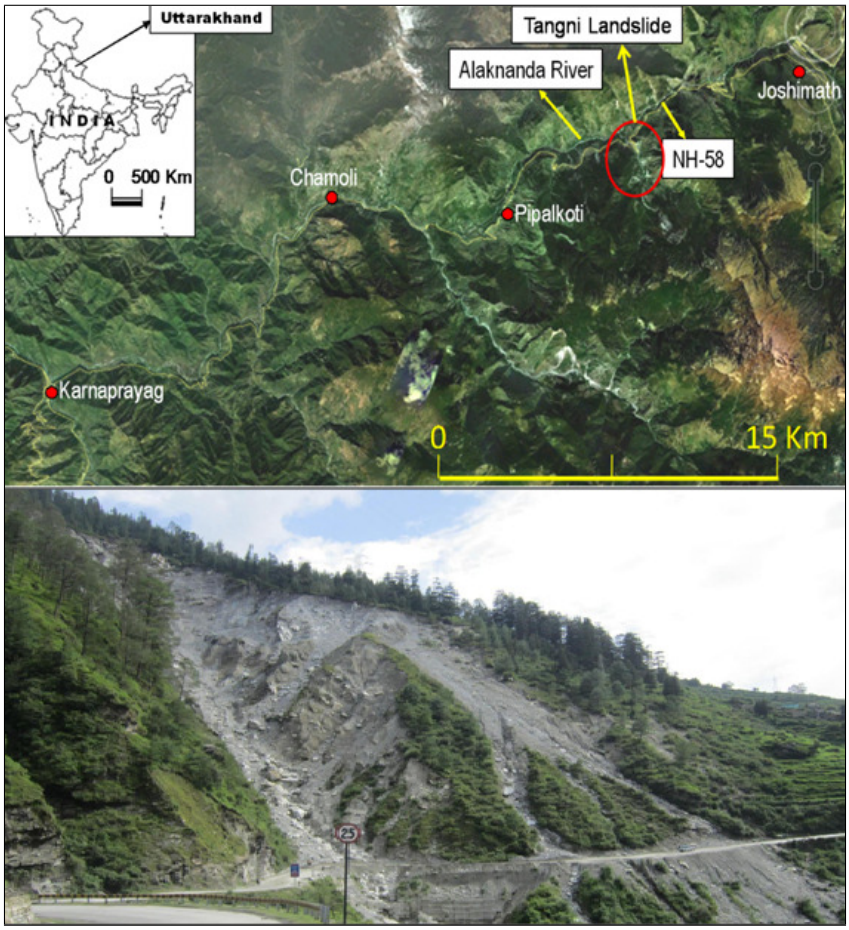

Figure 5.1 Tangni Landslide at Chamoli-Joshimath Corridor, Rishikesh-Badrinath Highway, Garhwal Himalaya, Uttarakhand [56]

There are many examples of the negative manifestations of landslides in various regions of India. India has witnessed ruinous landslides in the last few years. Listing from the year 2000, Amboori landslide Kerala, Kedarnath landslide Uttarakhand, Malin landslide Maharashtra are among the most destructive ones where thousands of people lost their lives and some also got missing. The landslide in Chamoli district of Uttarakhand is also causing huge discomfort and lose. Northwest Himalaya including Uttarakhand, Himachal Pradesh and Jammu & Kashmir has emerged as highest landslide hazard-prone

regions according to a [2]report submitted in the Parliament of India on July 27, 2016, by the Minister of State for Earth Sciences, Science and Technology. Bhagirathi valley in the Uttarkashi district, Uttarakhand (India) has 235 unstable zones, the largest number among all landslide-prone regions. Every year, landslides are generally triggered by heavy rainfall in this valley. Uttarakhand state has 24 areas spread in 344 locations including Bhagirathi valley and Kedarnath township which falls under very unstable zones. Here, the recurrent landslide problem has been witnessed over the years.

There are mainly two *causes* for a landslide. First is a natural (inevitable) cause which is beyond human control and order. Second is human triggered landslide caused by increasing pressure on the environment in the form of construction work, unlawful mining, heavy machine vibrations in ground and hill-cutting. All these mentioned activities can cause instability of the soil and slope surface which is the reason behind slope movement and displacement. Earthquakes, heavy rainfall, groundwater disbalance, soil erosion are a few factors which affect the stability of the slope naturally. Landslides might often lead to another natural calamity like floods to further worsen the living conditions.

Figure 5.1 shows the Tangni Landslide for the [3, 4] implementation of LEWS which is situated at Chamoli-Joshimath Corridor, Rishikesh-Badrinath Highway, Garhwal, Himalaya, Uttarakhand. Currently, the site is under preparation for the installation of the sensors.

5.1.2 Why LEWS?

Almost any area in the world can experience a landslide. Landslides cannot be stopped at times, be it a shallow landslide or large landslide. But the tremendous loss to the ecosystem can be somewhat brought down by employing Landslide Early Warning System (LEWS) which alerts the local people and administration, prior to the start of a landslide. Landslide monitoring is generally not practiced in India and an effective warning system is required to help the people get evacuated in such conditions.

Keeping in view the societal and strategic relevance of landslide disasters, the replication of the LEWS in landslide-prone states will help the respective State Disaster Management authorities in the landslide risk reduction. It will help the administration to prepare emergency services in advance and to plan out some strategy to lessen the threat. LEWS is a system which records and analyzes the slopes changing characteristics in real time and is able to predict the occurrence of landslides with various warning levels in real-time. Data flow diagram of a LEWS is given in figure 5.2. It shows how various parts of the LEWS are connected to each other and how the data flows in the system.

Figure 5.3 vividly demonstrates the hardware components and the architecture used for the development of LEWS for Tangni Landslide. It includes an IoT Node or a Coordinator Node with the ability

[2]https://timesofindia.indiatimes.com/city/dehradun/Ukhand-has-maximum-landslide-prone-areas-among-north-Himalayan-states-claims-min-of-earth-sciences-report/articleshow/53512570.cms

[3]NMHS funded project Evaluation and Design of Low-Cost Ground Instrumentation with Real Time Monitoring for the Development of Landslide Early Warning System; Project ID: NMHS/2017-18/MG47/31; Grant: 2.13 Cr.; Weblink: http://nmhs.org.in/MG_30_2017_18.php

[4]Author is working on the implementation of the project which involves the developments related to IoT Messaging Protocol, Database, Machine Learning based Data Analysis and Warning System and Web Interface implementation of LEWS.

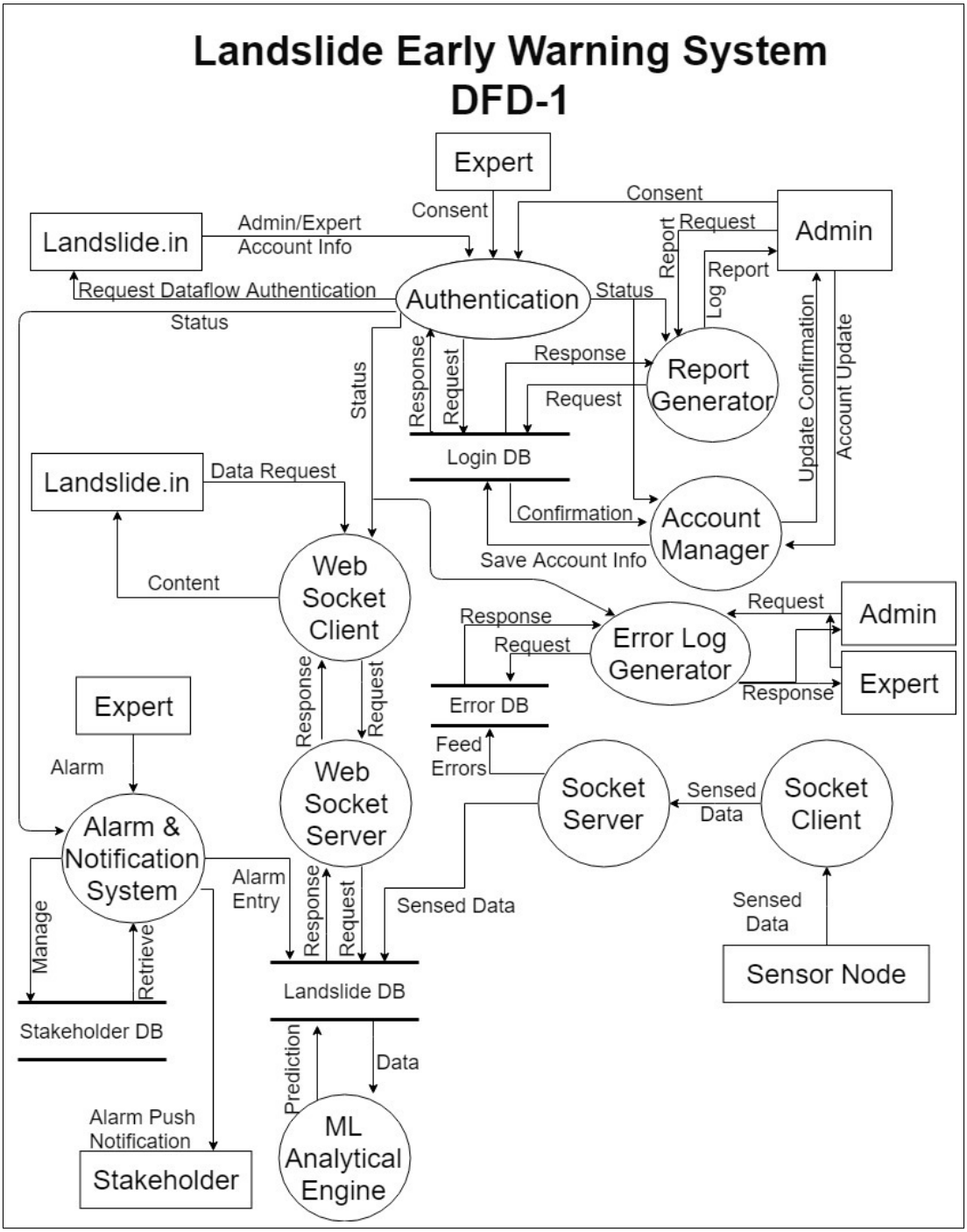

Figure 5.2 Data Flow in a LEWS

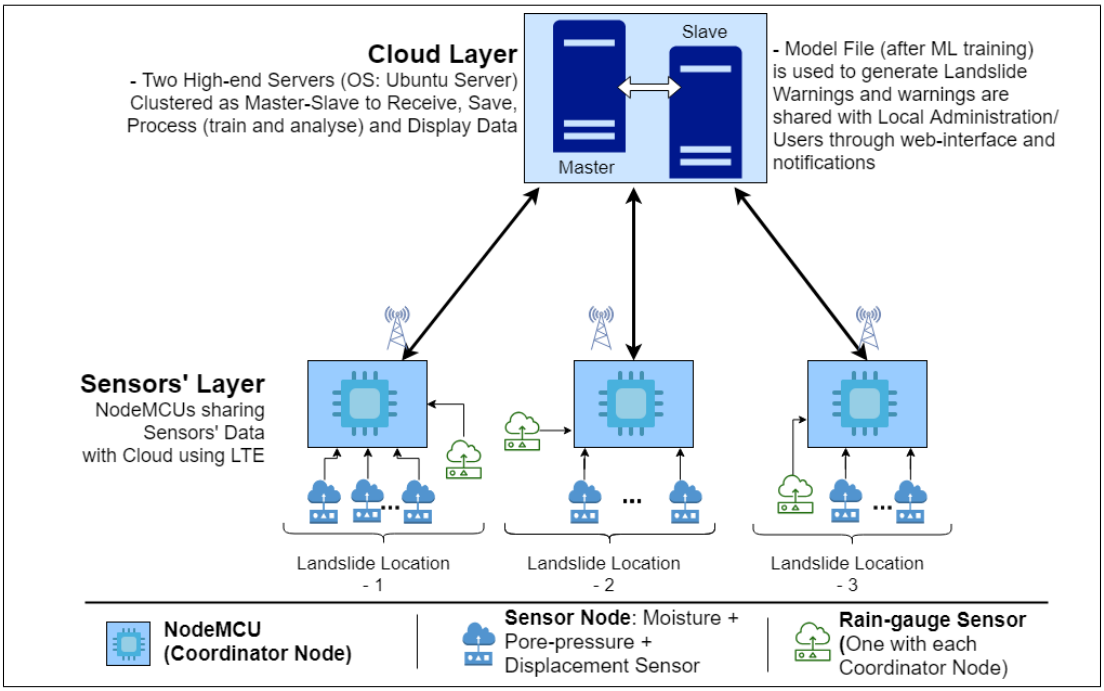

Figure 5.3 Hardware used to implement Basic LEWS

to transmit data through GSM/LTE. One Coordinator Node covers and represents a landslide location. It captures data from a number of Sensors Nodes and the number depends upon the size of the landslide as each node is installed at a distance of 30-50 Meters from each other. One Sensor Node is the collection of three types of sensors that are Displacement Sensor, Pore Pressure Sensor and Moisture Sensor. Each of the Coordinator Nodes holds a Rain-gauge Sensor for the entire landslide location as it can measure the rain intensity for a few miles. The Coordinator Node is connected to a Cloud/Central Server which can receive, store, analyze and display sensed data and notify the subscribers regarding a landslide situation.

5.1.3 Scope of LEWS

LEWS is focused on decreasing the impact of landslides in hilly areas. As landslide is a natural hazard and can not be stopped from happening in most of the cases, it is essential to warn people in its range. So, the scope of this research ranges from learning landslide process and available solution of landslide warning. Based on that, a *reliable and fault-tolerant* version of Landslide Early Warning System is under development and will be tested for different landslide locations.

5.1.4 Why Edge for LEWS?

LEWS requires high reliability and near real-time response for any emergency to alert nearby residents in time. In the case of a natural disaster like landslide or earthquake, there are high chances of the

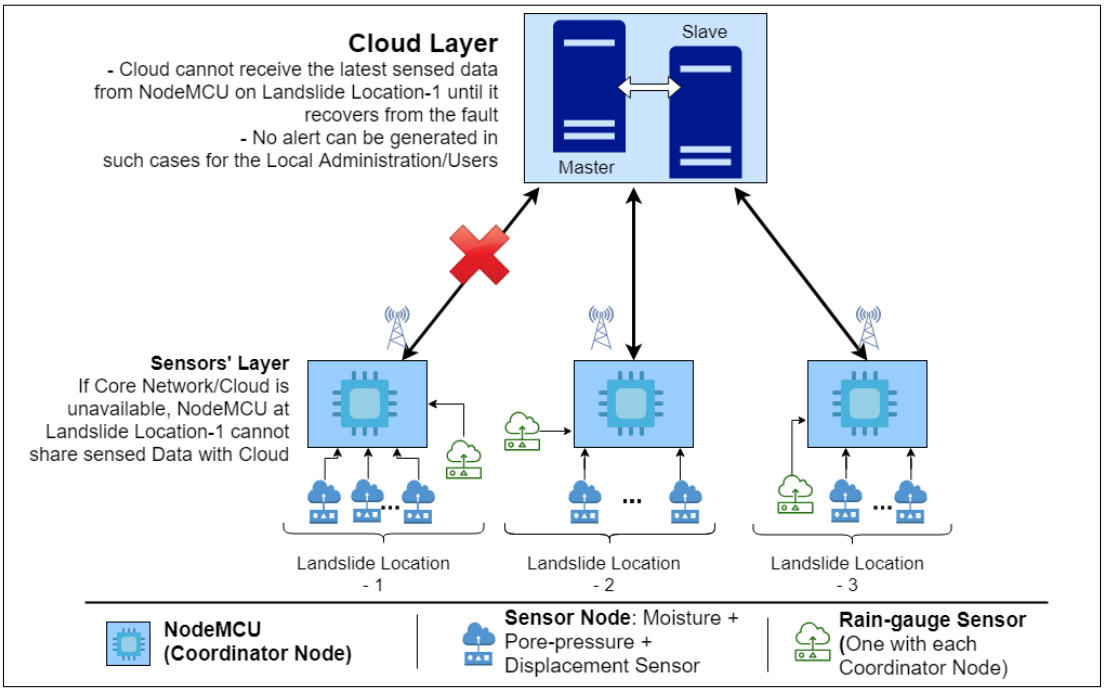

Figure 5.4 Why Edge Server is Important?

loss of network connectivity between sensors and server. Hence, to generate accurate early warnings for a possible landslide, computing resources are required to be up and running all the time. In the event of non-availability of computing resources, the system cannot predict the landslide event and fails. All the available landslide warning systems lack any mechanism to deal with Cloud failure or communication failure between IoT node and Cloud Server as shown in figure 5.4. If any such event happens, Coordinator Nodes can not share sensed data with the Cloud. Considering the computation power and storage capacity of usual Coordinator Nodes, neither they are able to store much of sensed data, nor they can process the data to generate a warning. Hence, the architecture is highly unreliable. Edge Computing is emerging as a complement to Cloud Computing to deal with such scenarios. It motivates to resolve connection failure and reliability issues.

Hence, LEWS is one of the perfectly tailored applications for the implementation of the proposed solution(s) in this thesis especially in Chapters 2 and 3.

5.2 Reliable LEWS using Edge Computing : Experimental Setup

LEWS is an IoT based landslide monitoring system which consists of data acquisition and analysis unit to predict the occurrence of landslides with various warning levels in near real-time. The proposed Reliable LEWS enrols Edge Computing paradigm in its architecture. Figure 5.5 demonstrates the block diagram and figure 5.6 demonstrates the architecture of LEWS with Edge Computing. The sensors data is transmitted to a Cloud Server from the *Coordinator Node* through an *Edge Server* in the middle.

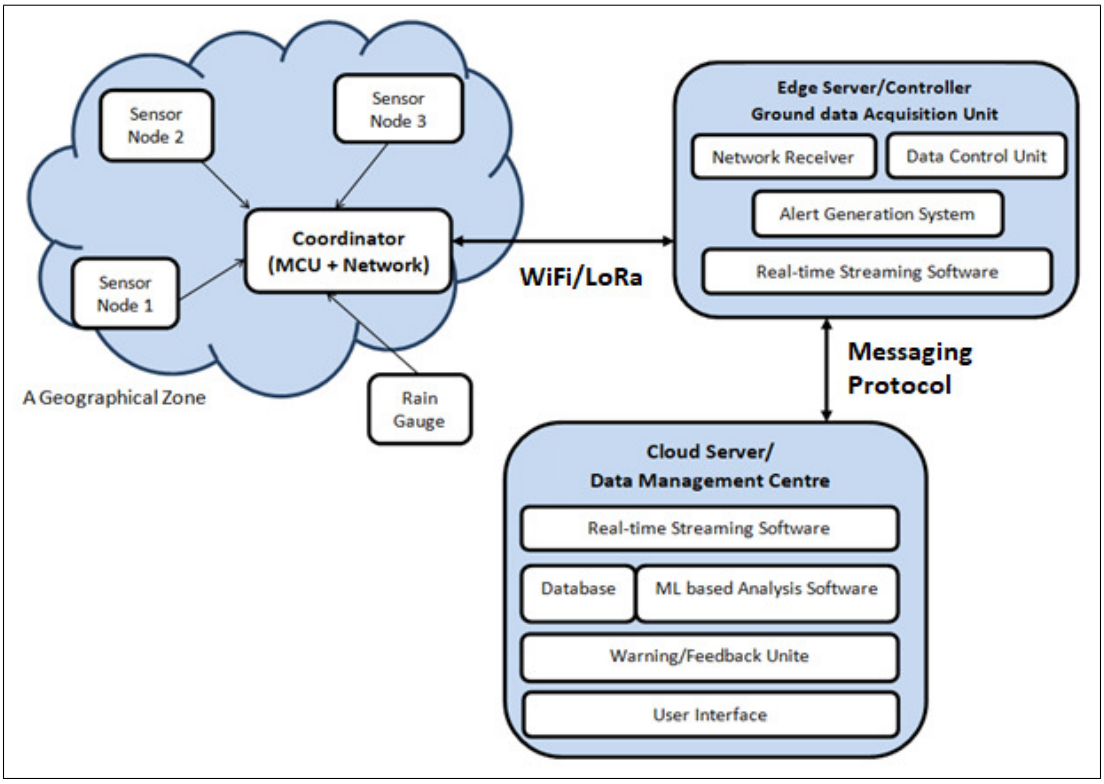

Figure 5.5 Block Diagram of LEWS with Edge Server

The proposed mechanism using Edge Server along with its working and importance for a Reliable LEWS is explained further in this section. Figure 5.7 exemplifies the proposed architecture for Reliable LEWS after the inclusion of Edge Server between Coordinator Node and Cloud Server. It also presents the control and computation devices used for the development of the system. It divides the entire architecture into three layers namely *Cloud Layer*, *Edge Layer* and *Extreme Edge/Sensors' Layer*. Further this section describes the functionality, devices/equipment and communication technologies used on these layers, as per the laboratory experimental setup.

5.2.1 Cloud Layer

Cloud Layer holds two high-end computer servers clustered together as master-slave. These servers are powered by [5]Ubuntu Server 18.04 LTS as an OS.

Task of *Master Server* is to receive and transmit data, to and from external devices respectively. It also hosts the landslide website to display sensed data trends. [6]Django framework is used to develop the landslide website.

[5]http://releases.ubuntu.com/18.04/
[6]https://www.djangoproject.com/

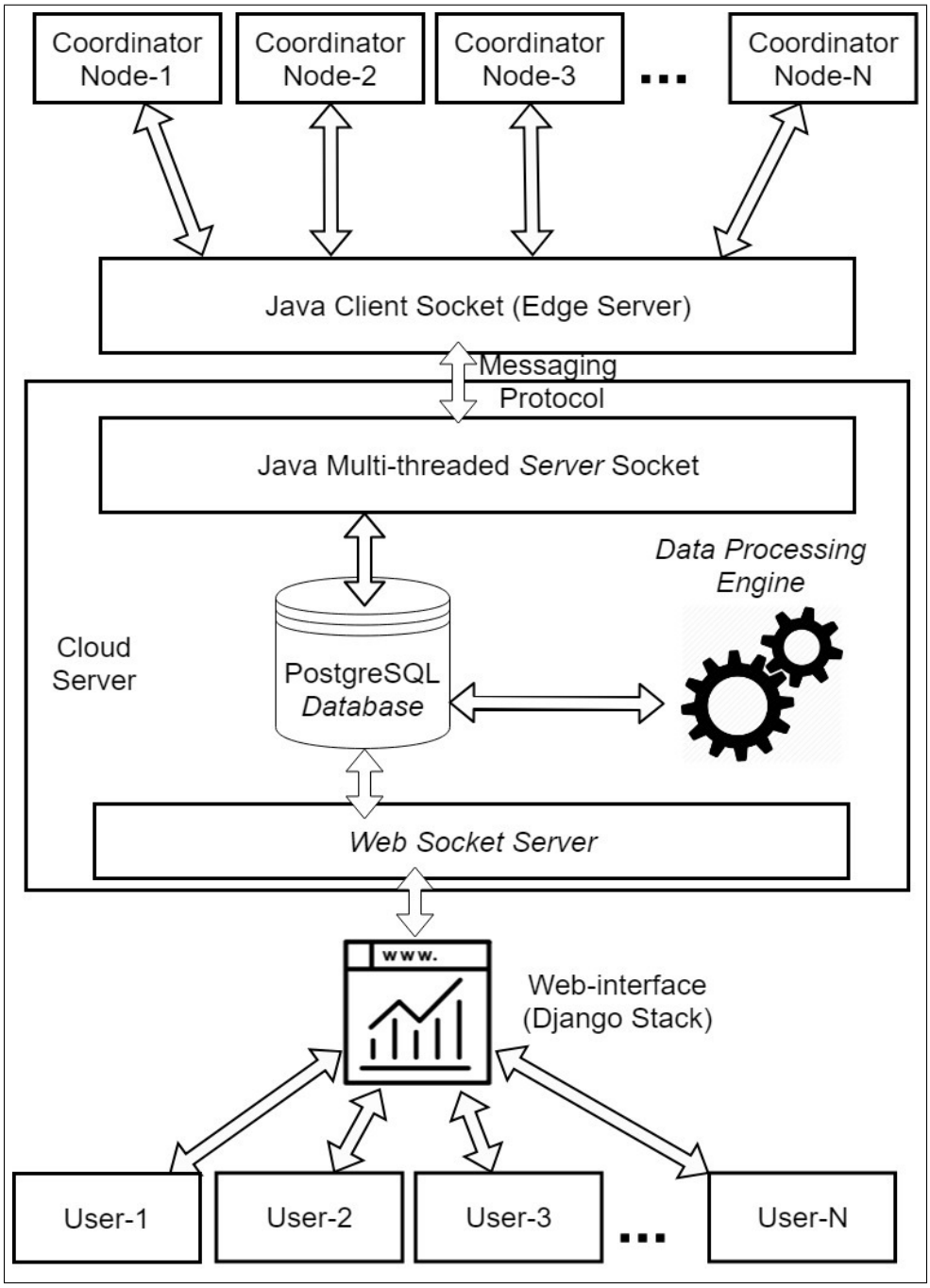

Figure 5.6 LEWS Architecture with Edge Server

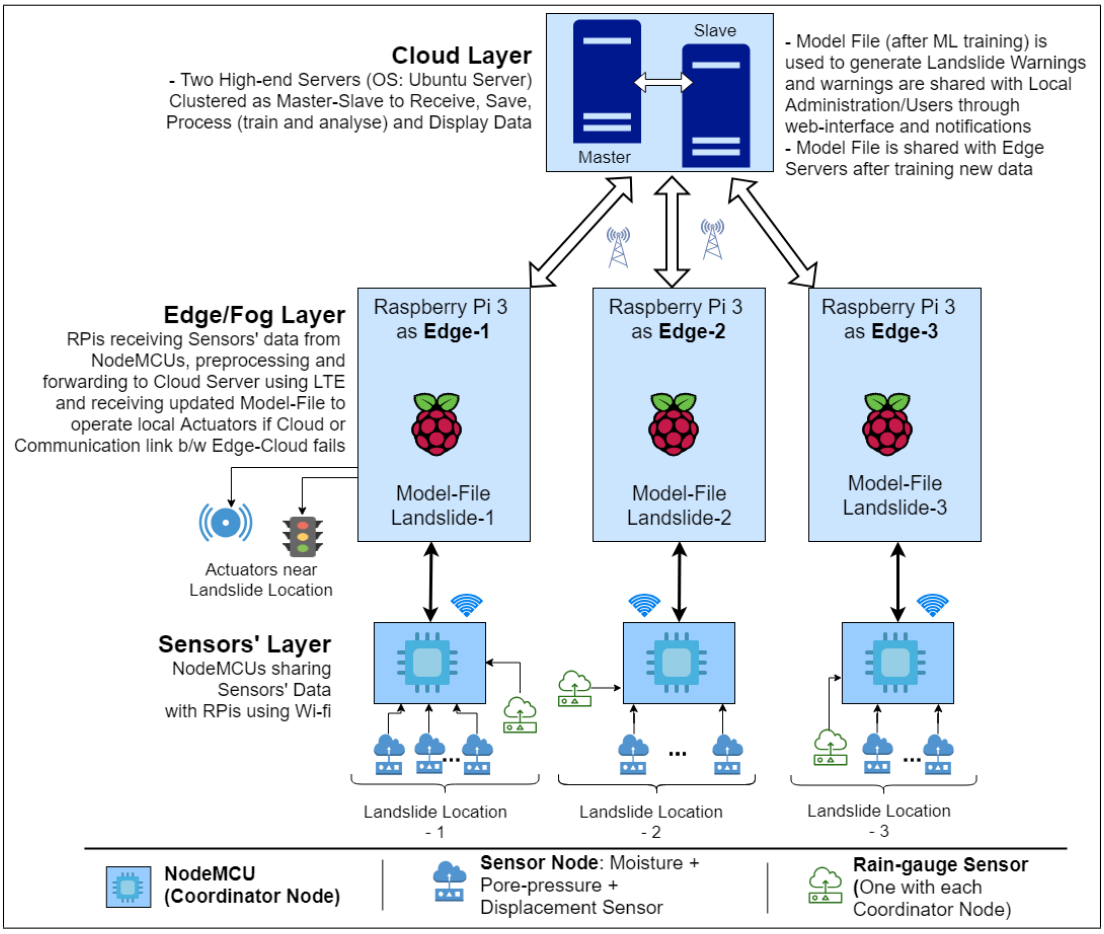

Figure 5.7 LEWS implementation with Edge Server

The major task of *Slave Server* involves storage and computation. It helps in securing data from unauthorized external access as only Master Server is accessible from outside. [7]PostgreSQL DB stores the sensed data. Major computation task involves training of sensed data for applying Machine Learning to generate landslide warnings. Model-file is created periodically after receiving new data. A fresh copy of The Model-file is shared with the Edge Server whenever Cloud Server generates it again.

Each of the servers is equipped with 4 Nvidia GTX 1080 TI 11GB GDDR5X GPUs, 128 GB RAM, AMD Ryzen Threadripper 1950X Processor, and ASRock X399 Gaming sTR4 SATA 6Gb/s USB 3.1/3.0 ATX AMD Motherboard. This configuration makes it capable of doing heavy processing in near real-time.

5.2.2 Edge/Fog Layer

The Edge Server has its own database and processing capacity. The complexity of services provided by the Edge Server is generally lower in comparison to the Cloud Server. It can be placed at any safe

[7]https://www.postgresql.org/

location near to the Coordinator Node, which is less prone to network and connection failures. The Edge Server on a landslide location can be located a few metres to a few kilometres away depending upon the availability of a secure location from where uninterrupted communication can be established between the Edge and the Cloud. Nearby villages or residential areas are some of the most appropriate locations for deploying an Edge Server. Edge mostly has lesser computation resources while compared to the Cloud Server but can generate alert/warning quicker than the Cloud. It is feasible because of the reduction of communication cost during data transmission through the core network. Even though the Edge Server is unable to train ML models using the sensed data, it is able to utilize the Cloud generated model(s) to analyze recent data and initiate warnings for the nearby areas. If the communication between the Edge and the Cloud is lost, still, the system stays operational due to the Edge Server.

To create an Edge Server, 3 Raspberry Pi [8](RPi) 3B+ are used with WLAN, Ethernet facility and ability to connect SIM900A GSM to transmit data to the Cloud Server. RPi receive sensors' data from the Sensors' Layer and after prepossessing forward it to the Cloud Server. It can trigger a number of actuators for the nearby residents such as alarm or traffic lights in case of a landslide situation. It stores the Modelfile shared by the Cloud Server for generating landslide warning.

5.2.3 Extreme Edge/Sensors' Layer

Extreme Edge or Sensors' Layer is implemented using [9]NodeMCU that collects data from the Sensor Nodes and a Rain-gauge sensor located on landslide location. Though, a Sensor Node is the combination of 3 sensors that are Moisture, Pore-pressure and Displacement Sensors, a dummy data generator is used during experimentation as these sensors are yet to be deployed on the landslide for our project. There can be more than one Coordinators connected to the Edge Server but during experimentation 1 NodeMCU is connected to 1 RPi assuming that each landslide has a coordinator and a dedicated Edge Server as RPi.

5.2.4 Communication and Data Transmission Technologies

To transmit data between the Edge Server and the Cloud Server, GSM is used for communication as it is one of the most suitable options for landslide areas. The RPis' built-in wi-fi is used for the experiment for transmitting sensed data from the Coordinator Node on Sensors' Layer.

As far as communication protocols are concerned, the MQTT protocol is used for real-time streaming of data between NodeMCU at Sensors' Layer and RPi at Edge Layer. For data transmission between RPi and Cloud Server, Java Client-Server Socket is implemented.

The codes on NodeMCU and RPi holds the conditions to test the connects and alternate actions in case of connection fails. For instance, NodeMCU starts searching for an alternate wi-fi connection to transmit data to the Cloud. For security, RPis matches a key before allowing any new device to connect.

[8]https://www.raspberrypi.org/documentation/configuration/
[9]https://www.nodemcu.com/index_en.html

Similarly, if there is no connection between RPi and Cloud, RPi triggers a python script to access the Model-file and analyze the incoming sensors' data.

5.3 Laboratory Experiments : Edge Reliability Test

5.3.1 Case 1: Link Failure between Edge and Cloud

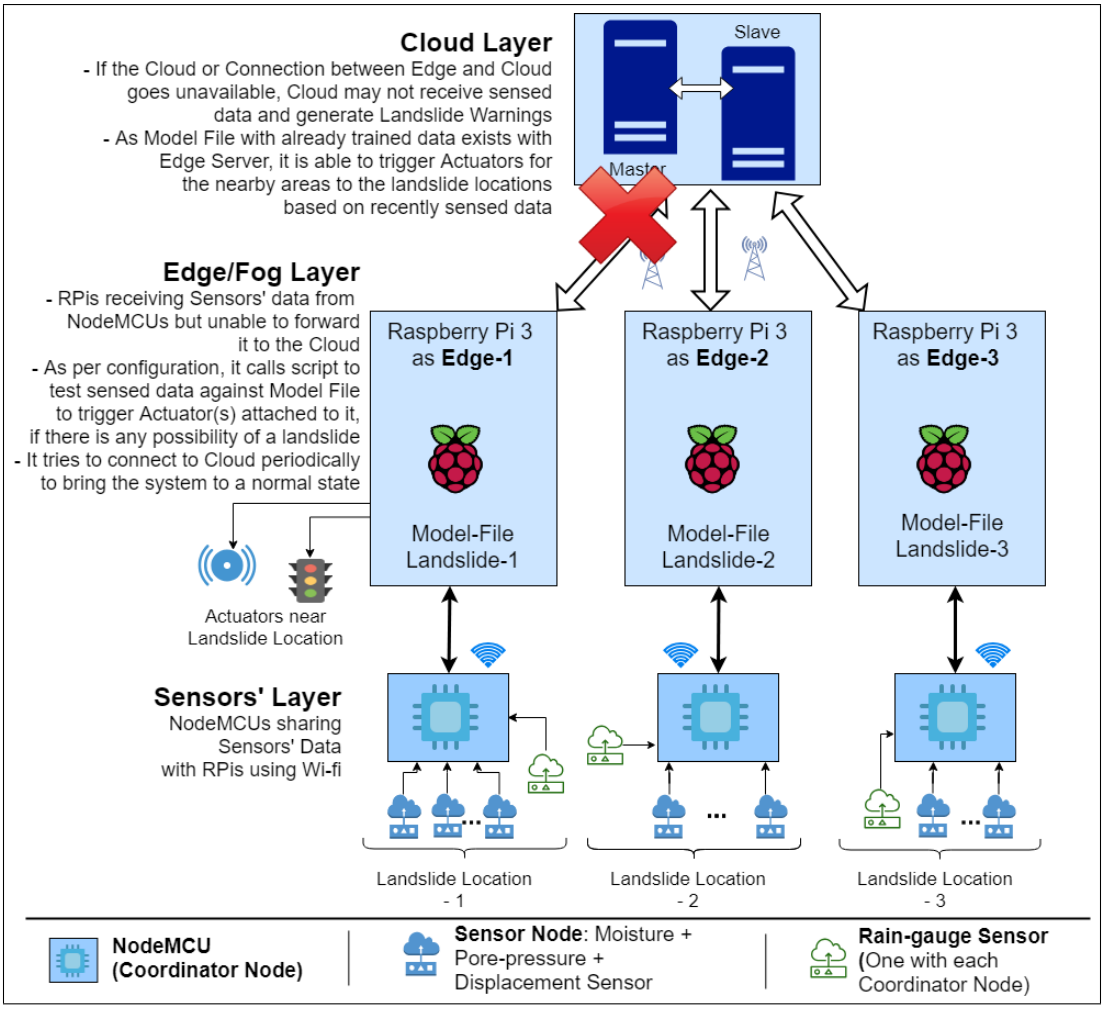

Figure 5.8 LEWS implementation if Cloud Fails or Communication between Edge and Cloud Fails

When the link between the RPi on Edge and the Cloud Server breaks due to any kind of error then RPi works as the mini-Cloud and completes the processing needs (refer figure 5.8). RPi can manage light computations, but complex algorithms running on the Cloud such as data training cannot run on RPi due to its resource constraints.

Cloud continuously runs training and testing procedures on the sensed data, it receives. During ML-based data training at the Cloud, a Model-file is generated. Cloud refreshes this Model-file on RPi,

whenever a new one is available. This file is extremely crucial to analyze data in the absence of the Cloud. So, whenever the Cloud goes down or the connection between RPi and the Cloud breaks, RPi utilizes the latest Model-file to process the incoming data from NodeMCU at a lower layer. If after processing, RPi finds an anomaly, it raises an alarm through actuator for the people in its locality and further actions can be taken. When the connection to the Cloud is regained then the procedure will be back to normal and processing will be carried in Cloud. In this way, even if the services from the Cloud are not accessible, an Edge Server is up and running for desired computations.

5.3.2 Case 2: Link Failure between Edge and Coordinator

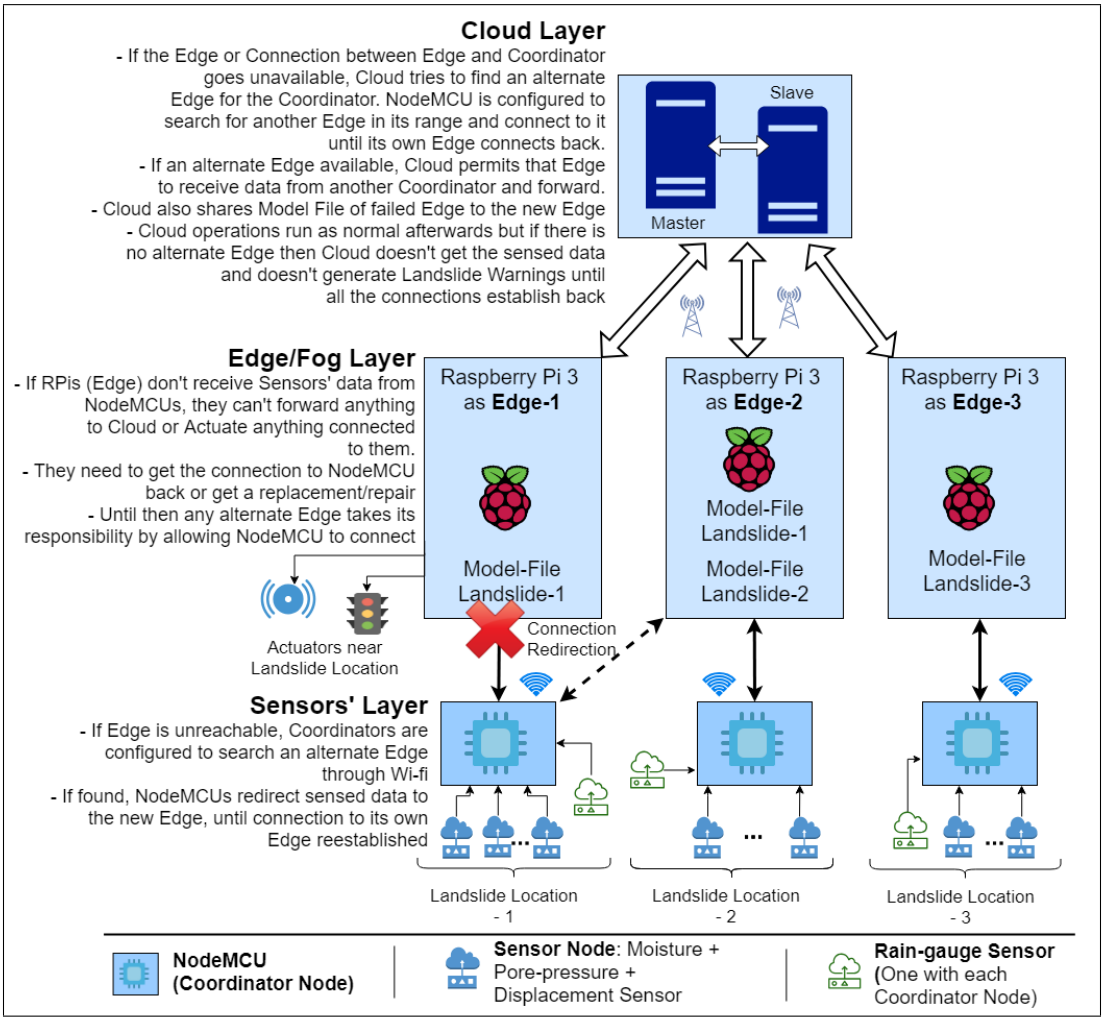

Figure 5.9 LEWS implementation if Edge Fails or Communication between Edge and Coordinator Fails

The next and bigger thing to worry about is what would happen when the Edge Server itself fails or the link between the Edge and Coordinator gets broken. One of such scenarios is shown in figure 5.9.

To ensure the reliability at Edge level, the concept of switching and replication is introduced in the system. As described in the experimental setup, there is an RPi for each of the 3 landslide locations, which collects the data from the NodeMCU located at that landslide. It is been assumed that for a landslide location, other RPi(s) are in range as an alternate Edge Server. If that is not the case and no alternate Edge is in range then this scenario is not different than the one shown in figure 5.4 where the Coordinator node was the IoT node and if that node fails, the system fails. But, with the proposed mechanism, that is the worst case scenario and the overall system is considerably beneficial for the LEWS.

Considering the availability of an alternate RPi in range of the NodeMCU for which its own RPi is unavailable, it can switch its connection to an alternate RPi through wi-fi scanning and pairing. After the completion of switching between NodeMCU and alternate RPi, RPi will receive and forward data from two landslide locations, until the old connections resume. Replication takes place as Cloud shares the Model-file of the newly connected landslide location to the alternate RPi. Now, this RPi can trigger actuators for both the landslide locations, it is connected to. Of course, the proposed switching and replication mechanism is an overhead for the system performance. However, for keeping the system reliable and running all the time, a small degradation in the system's performance is acceptable for LEWS.

5.4 Possible Results: 3 Landslide Locations

During experimentation, it was assumed that there are alternatives available at Edge Layer. That makes the system availability almost 100%. But, this may not be the case all the time. This section is dedicated to theoretically demonstrate all the possible scenarios with 3 landslide locations.

The reliability analysis consists of multiple cases which are based on the number of Edge Nodes' failure at a particular instance. In our experimental setup, we considered the 3 Edge Nodes to offer the seamless transmission of the sensors' data to the Cloud Servers. Our analysis recognizes four possible cases based on the number of Edge Nodes' failures.

Table 5.1 Cases and respective Number of Edge Node(s) Failures

Case	No. of Edge Node Fails
1	0
2	1
3	2
4	3

Evaluations of each case are discussed as follows:

5.4.1 Case: 1

In this case, all the 3 Edge Nodes are always available without failure and the user gets access to the working Edge Node in maximum 1 trial.

Number of Edge Node Fails = 0

Number of up/available Edge Nodes = 3

Number of down/unavailable nodes = 0

Total Number of Trials = 1

Number of Failed Trials = 0

Now, we find out the ability to transfer the information or to perform a specific task even in case of failure of some edges/components.

The probability of availability (A_t) is defined as

$$\text{Probability of Availability } (A_t) = \frac{\text{Number of up Edge Nodes}}{(\text{Number of up Edge Nodes + Number of down Edge Nodes})} \tag{5.1}$$

$$\text{So, } A_t = \frac{3}{(3+0)} = 1$$

The failure frequency is used to measure the reliability of the proposed system. Failure probability is defined as:

$$\text{Failure Probability } (F_p) = \frac{\text{Number of failed trials}}{\text{Total number of trials}} \tag{5.2}$$

$$\text{So, } F_p = \frac{0}{1} = 0$$

The failure frequency is used to measure the reliability of the proposed system. Reliability is defined as:

$$\text{Probability of Reliability } (R) = 1 \quad \text{Failure Probability}(F_p) \tag{5.3}$$

$$\text{So, } R = 1 \quad F_p = 1 \quad 0 = 1$$

5.4.2 Case: 2

In this case, two Edge Nodes are always available without failure then the user gets access to the working Edge Node in maximum 2 trials.

Number of Edge Node Fails = 1

Number of up/available Edge Nodes = 2

Number of down/unavailable nodes = 1

$$\text{Probability of Availability, } (A_t) = \frac{2}{(2+1)} = 0.66666667 \tag{5.4}$$

5.4.2.1 Case: 2-a

If the Total Number of Trials = 1 then the Number of Failed Trials = 0

$$\text{Failure probability, } (F_p) = \frac{0}{1} = 0 \tag{5.5}$$

$$\text{Probability of Reliability, } (R) = 1 \quad F_p = 1 \quad 0 = 1 \tag{5.6}$$

5.4.2.2 Case: 2-b

If the Total Number of Trials = 2 then the Number of Failed Trials = 1

$$\text{Failure probability, } (F_p) = \frac{1}{2} = 0.5 \tag{5.7}$$

$$\text{Probability of Reliability, } (R) = 1 \quad F_p = 1 \quad 0.5 = 0.5 \tag{5.8}$$

5.4.3 Case: 3

In this case, one Edge Node is always available without failure then the user gets access of working Edge Node in maximum 1 trial.

Number of Edge Node Fails = 2

Number of up/available Edge Nodes = 1

Number of down/unavailable nodes = 2

$$\text{Probability of Availability, } (A_t) = \frac{1}{(1+2)} = 0.33333333 \tag{5.9}$$

5.4.3.1 Case: 3-a

If the Total Number of Trials = 1 then the Number of Failed Trials = 0

$$\text{Failure probability, } (F_p) = \frac{0}{1} = 0 \tag{5.10}$$

$$\text{Probability of Reliability, } (R) = 1 \quad F_p = 1 \quad 0 = 1 \tag{5.11}$$

5.4.3.2 Case: 3-b

If the Total Number of Trials = 2 then the Number of Failed Trials = 1

$$\text{Failure probability, } (F_p) = \frac{1}{2} = 0.5 \tag{5.12}$$

$$\text{Probability of Reliability, } (R) = 1 \quad F_p = 1 \quad 0.5 = 0.5 \tag{5.13}$$

5.4.3.3 Case: 3-c

If the Total Number of Trials = 3 then the Number of Failed Trials = 2

$$\text{Failure probability, } (F_p) = \frac{2}{3} = 0.66666667 \tag{5.14}$$

$$\text{Probability of Reliability, } (R) = 1 \quad F_p = 1 \quad 0.66666667 = 0.3333333 \tag{5.15}$$

5.4.4 Case: 4

In this case, all the Edge Nodes are unavailable then the user unable to access the Edge Node.

Number of Edge Node Fails = 3

Number of up/available Edge Nodes = 0

Number of down/unavailable nodes = 3

$$\text{Probability of Availability, } (A_t) = \frac{0}{(0 + 3)} = 0 \tag{5.16}$$

If the Total Number of Trials = 3 then the Number of Failed Trials = 3

$$\text{Failure probability, } (F_p) = \frac{3}{3} = 1 \tag{5.17}$$

$$\text{Probability of Reliability, } (R) = 1 \quad F_p = 1 \quad 1 = 0 \tag{5.18}$$

The availability and reliability in all the discussed cases has been observed and the corresponding results are summarized in Table 5.2

Figure 5.10 pictorially summaries the comparison of Availability and Reliability for all the possible cases with 3 Edge Servers used in experimentation. Case 1 achieves the highest probability of availability and reliability in the experimental setup. Also, average availability i achieved in Case 2 with a probability of reliability up to 1. The worst case scenario is presented in Case 4 when Edge Nodes are unavailable for seamless transmission of the sensors' data to the Cloud Server.

Table 5.2 Availability and Reliability Matrix for all the possible Cases with 3 Edge Servers

Case	No. of Edge Node Fails	Up Nodes	Down Nodes	Total No. of Trials	Number of Failed Trials	Availability	Failure probability	Reliability
1	0	3	0	1	0	1	0	1
2a	1	2	1	1	0	0.66666667	0	1
2b	1	2	1	2	1	0.66666667	0.5	0.5
3a	2	1	2	1	0	0.33333333	0	1
3b	2	1	2	2	1	0.33333333	0.5	0.5
3c	2	1	2	3	2	0.33333333	0.66666667	0.3333333
4	3	0	3	3	3	0	1	0

5.5 Summary

The proposed Reliable LEWS using Edge Computing is expected to convert the whole IoT Cloud architecture as reliable as possible. Even if the connection to Cloud Server would be down for a certain period of time, the Edge Server would be readily available for the processing the current incoming sensed data from the Coordinator Node and raise warning alarms, if required. The system is also expected to handle Edge Server failure by introducing replication of the Model-files after training and link switching from the failed Edge to an alternate nearest available Edge Server. In this way, the computation of the sensed data takes place both at the Cloud and the Edge, according to the circumstances. Hence, it increases the LEWS's reliability.

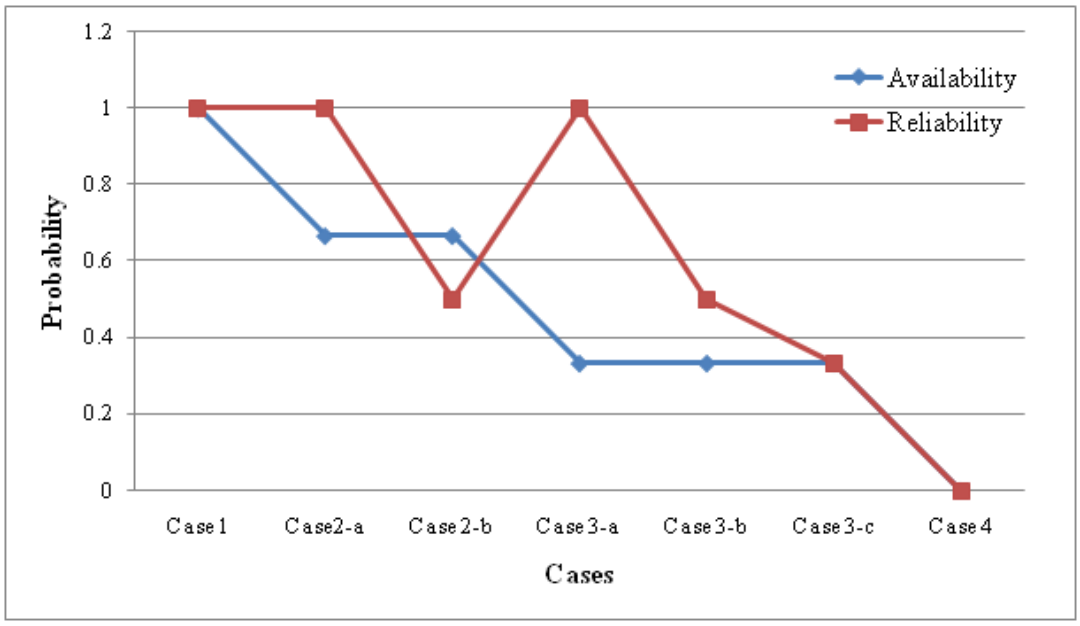

Figure 5.10 Comparison of Availability and Reliability for all the possible Cases with 3 Edge Servers

Chapter 6

Conclusions

This chapter, concludes this thesis with a summary of contributions and the directions for the future researches.

6.1 Conclusion

Evolution of Cloud Computing facilitated the development of N number of IoT applications in various sectors. Many of the recent applications are delay sensitive with high requirements of reliability. So, latency, reliability and fault-tolerance are some of the major concerns in the IoT-Cloud infrastructure primarily for the delay sensitive IoT applications. To resolve these issues, solutions are divided into three parts which are focused on providing reliable services to the IoT applications when either Cloud resources are unavailable, Edge resources are unavailable or applications require resource allocation at the edge in dynamic environment.

The first part provides a solution to real-time IoT applications with low latency and high reliability requirements. A hierarchical Edge-Cloud architecture is presented as an advancement of current IoT architecture. With the inclusion of multiple layers of processing, applications keep on providing services to the users even if the resources at higher layers either gets fail or disconnect. Newly introduced layers work on the edge or the extreme edge of the network and provide near real-time responses to the IoT devices. So, Edge Computing fixes the issues related to unavailability of high layer computation resources and latency.

After introducing edge devices with the Cloud, IoT applications can use Edge Servers to get feedback. It improves the reliability of the application if the Cloud gets disconnected. However, there can be a case that any of the serving edge devices may also get unavailable. So, the second part deal with the reliability and fault-tolerance at the edge of the network. This part ensures that the system is capable to deal with Edge Servers' failures. A switching and replication mechanism is used to sort this situation out. This is helpful in searching and reconnecting to an alternate Edge Server, if available. Simultaneously, Cloud shares important information of the IoT application to the alternate serving Edge Server, if required.

The third part presents an Edge Controller that allocates edge computational resources to a large number of IoT applications in a reliable manner. Due to the unavailability of enough edge resources, privately owned or any other type of devices are considered to be used, if available. As privately owned devices are highly dynamic in nature, Edge Controller needs to allocate such devices reliably to the IoT applications. It is done by prioritizing the available edge resources for the IoT applications.

The proposed reliable and fault-tolerant architecture using Edge Computing is implemented as a part of Landslide Early Warning System. The efficiency and utility of the proposed mechanisms have been proved using theoretical analysis and simulations too.

6.2 Future perspectives

As an extension, there is scope to investigate this research work further. Edge Controller has the potential to be the part of Edge Computing in the near future. Though, its simulation proved its worth, its real life implementation is required to be done to test its performance and applicability. On the other hand, until now Edge Servers have been created using existing general purpose devices such as RPis. To improve the performance of IoT applications using Edge, research is required to be done to develop application specific edge devices with inbuilt fault tolerance mechanisms.

Publications

1. Jitender Grover and Ram Murthy Garimella, "Concurrency and Synchronization in Structured Cyber Physical Systems", book chapter in CyberPhysical Systems: Architecture, Security and Application, EAI/Springer Innovations in Communications and Computing, Editors: Song Guo, The Hong Kong Polytechnic University; Hong Kong & Deze Zeng, China University of Geosciences, China, pp. 73-99, 2019. doi. https://doi.org/10.1007/978-3-319-92564-6_5

2. Jitender Grover and Ram Murthy Garimella, "Optimized Edge Computing and Small-Cell Networks", book chapter in Edge Computing: From Hype to Reality, EAI/Springer Innovations in Communications and Computing, Editors: Fadi Al-Turjman, Middle East Technical University, Turkey, pp. 17-31, Nov 2018. doi. https://doi.org/10.1007/978-3-319-99061-3_2

3. Jitender Grover, Garimella Rama Murthy, Reliable and Fault Tolerant IoT-Edge Architecture, IEEE SENSORS 2018 (a flagship conference of IEEE Sensors Council), Pullman Aerocity, New Delhi, India, October 28-31, 2018.

4. Jitender Grover, Rhishi Pratap Singh, Garimella Rama Murthy, Reliability based Resource Allocation in Software Defined Edge Controller for IoT Infrastructure, Poster in IEEE SENSORS 2018 (a flagship conference of IEEE Sensors Council), Pullman Aerocity, New Delhi, India, October 28-31, 2018.

5. Rhishi Pratap Singh, Jitender Grover, Rama Murthy Garimella, Software Defined Cognitive Edge Controller in IoT Infrastructure, submitted in International Journal of Network Management, ISSN: 1099-1190.

6. Rhishi Pratap Singh, Jitender Grover, Garimella Rama Murthy, Self Organizing Software Defined Edge Controller in IoT Infrastructure, International Conference on Internet of Things and Machine Learning (IML17), Article No. 31, doi: 10.1145/3109761.3158390, pp. 1-7, Liverpool John Moores University, United Kingdom, October 17-18, 2017.

7. Gadiraju Divija Swetha, Jitender Grover, Garimella Rama Murthy, Dynamic Channel Allocation in Small Cells, IEEE 7th International Conference On Reliability, Infocom Technologies And Optimization (ICRITO'2018), Amity University, Noida, India, August 29-31, 2018.

Bibliography

[1] J. Wan and M. Xia, "Cloud-assisted cyber-physical systems for the implementation of industry 4.0," *Mobile Networks and Applications*, vol. 22, no. 6, pp. 1157–1158, 2017.

[2] "Cisco visual networking index: Forecast and trends, 20172022," White Paper, Cisco, Feb. 2019.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[6] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of everything*. Springer, 2018, pp. 103–130.

[7] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[8] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015.

[9] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[10] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[11] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, June 2015.

[12] A. M. Khan and F. Freitag, "On participatory service provision at the network edge with community home gateways," *Procedia Computer Science*, vol. 109, pp. 311–318, 2017.

[13] S. Kim, "Nested game-based computation offloading scheme for mobile cloud IoT systems," *EURASIP Journal on Wireless Communications and Networking*, vol. 2015, no. 1, p. 229, 2015.

[14] F. Samie, V. Tsoutsouras, S. Xydis, L. Bauer, D. Soudris, and J. Henkel, "Distributed qos management for internet of things under resource constraints," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2016 International Conference on*. IEEE, 2016, pp. 1–10.

[15] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[16] T. X. Tran and D. Pompili, "Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks," *arXiv preprint arXiv:1705.00704*, 2017.

[17] Q. Fan and N. Ansari, "Cost aware cloudlet placement for big data processing at the edge," in *2017 IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[18] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A software defined networking architecture for the internet-of-things," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–9.

[19] F. Slim, F. Guillemin, and Y. Hadjadj-Aoul, "On virtual network functions' placement in future distributed edge cloud," in *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, Sept 2017, pp. 1–4.

[20] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[21] G. R. Murthy and R. P. Singh, "Time optimization in spectrum sensing: Interesting cases," in *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. IEEE, 2017, pp. 487–492.

[22] S. K. Datta, C. Bonnet, and J. Haerri, "Fog computing architecture to enable consumer centric internet of things services," in *2015 International Symposium on Consumer Electronics (ISCE)*. IEEE, 2015, pp. 1–2.

[23] M. Amoon, "A framework for providing a hybrid fault tolerance in cloud computing," in *2015 Science and Information Conference (SAI)*. IEEE, 2015, pp. 844–849.

[24] T. J. Charity and G. C. Hua, "Resource reliability using fault tolerance in cloud computing," in *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*. IEEE, 2016, pp. 65–71.

[25] N. Mohamed and J. Al-Jaroodi, "A collaborative fault-tolerant transfer protocol for replicated data in the cloud," in *2012 International Conference on Collaboration Technologies and Systems (CTS)*. IEEE, 2012, pp. 203–210.

[26] Y. Wang and Y. Pan, "Cloud-dew architecture: realizing the potential of distributed database systems in unreliable networks," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2015, pp. 85–89.

[27] S. Malik and F. Huet, "Adaptive fault tolerance in real time cloud computing," in *2011 IEEE World Congress on Services*. IEEE, 2011, pp. 280–287.

[28] N. Ivaki, S. Boychenko, and F. Araujo, "A fault-tolerant session layer with reliable one-way messaging and server migration facility," in *2014 IEEE 3rd Symposium on Network Cloud Computing and Applications (ncca 2014)*. IEEE, 2014, pp. 75–82.

[29] Z. Amin, H. Singh, and N. Sethi, "Review on fault tolerance techniques in cloud computing," *International Journal of Computer Applications*, vol. 116, no. 18, 2015.

[30] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," in *2013 20th International Conference on Systems, Signals and Image Processing (IWSSIP)*. IEEE, 2013, pp. 43–46.

[31] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli, "On qos-aware scheduling of data stream applications over fog computing infrastructures," in *2015 IEEE Symposium on Computers and Communication (ISCC)*. IEEE, 2015, pp. 271–276.

[32] K. C. Okafor, I. E. Achumba, G. A. Chukwudebe, and G. C. Ononiwu, "Leveraging fog computing for scalable iot datacenter using spine-leaf network topology," *Journal of Electrical and Computer Engineering*, vol. 2017, 2017.

[33] A. Martin, C. Fetzer, and A. Brito, "Active replication at (almost) no cost," in *2011 IEEE 30th International Symposium on Reliable Distributed Systems*. IEEE, 2011, pp. 21–30.

[34] M. Shen, A. D. Kshemkalyani, and T.-Y. Hsu, "Causal consistency for geo-replicated cloud storage under partial replication," in *2015 IEEE International Parallel and Distributed Processing Symposium Workshop*. IEEE, 2015, pp. 509–518.

[35] P. Skobelev and D. Trentesaux, "Disruptions are the norm: cyber-physical multi-agent systems for autonomous real-time resource management," in *International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*. Springer, 2016, pp. 287–294.

[36] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[37] M. M. Gaber, J. B. Gomes, and F. Stahl, "Pocket data mining," *Big Data on Small Devices. Series: Studies in Big Data*, 2014.

[38] Y. Ai, M. Peng, and K. Zhang, "Edge cloud computing technologies for internet of things: A primer," *Digital Communications and Networks*, 2017.

[39] P. Patil, A. Hakiri, and A. Gokhale, "Cyber foraging and offloading framework for internet of things," in *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, vol. 1. IEEE, 2016, pp. 359–368.

[40] A. Bhattcharya and P. De, "Computation offloading from mobile devices: Can edge devices perform better than the cloud?" in *Proceedings of the Third International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*. ACM, 2016, pp. 1–6.

[41] F. Samie, V. Tsoutsouras, L. Bauer, S. Xydis, D. Soudris, and J. Henkel, "Computation offloading and resource allocation for low-power IoT edge devices," in *Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on*. IEEE, 2016, pp. 7–12.

[42] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[43] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases & Future Directions," *IEEE Communications Surveys & Tutorials*, 2017.

[44] C. Kharkongor, T. Chithralekha, and R. Varghese, "A SDN Controller with Energy Efficient Routing in the Internet of Things (IoT)," *Procedia Computer Science*, vol. 89, pp. 218–227, 2016.

[45] Zach Shelby, "Constrained RESTful Environments (CoRE) Link Format," Internet Requests for Comments, RFC Editor, RFC 6690, August 2012. [Online]. Available: https://tools.ietf.org/rfc/rfc6690.txt

[46] C. Perkins and H. Harjono, *Resource Discovery Protocol for Mobile Computing*. Boston, MA: Springer US, 1996, pp. 219–236. [Online]. Available: https://doi.org/10.1007/978-0-387-34980-0_22

[47] C. Cabrera, A. Palade, and S. Clarke, "An evaluation of service discovery protocols in the internet of things," in *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 469–476.

[48] J. Quevedo, C. Guimarães, R. Ferreira, D. Corujo, and R. L. Aguiar, "ICN as Network Infrastructure for Multi-Sensory Devices: Local Domain Service Discovery for ICN-based IoT Environments," *Wireless Personal Communications*, pp. 1–20, 2017.

[49] H. Bryhni, E. Klovning, and O. Kure, "A comparison of load balancing techniques for scalable web servers," *IEEE network*, vol. 14, no. 4, pp. 58–64, 2000.

[50] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[51] , "intlinprog," https://in.mathworks.com/help/optim/ug/intlinprog.html.

[52] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[53] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

[54] , "gamultiobj," http://in.mathworks.com/help/gads/gamultiobj.html.

[55] S. P. Pradhan, V. Vishal, and T. N. Singh, *Landslides: Theory, Practice and Modelling*. Springer, 2019, vol. 50.

[56] D. P. Kanungo and S. Sharma, "Rainfall thresholds for prediction of shallow landslides around chamoli-joshimath region, garhwal himalayas, india," *Landslides*, vol. 11, no. 4, pp. 629–638, 2014.