BACKUP ONE HOP ROUTING OPTIMIZATION

by

SIDDARDHA KAJA

Thesis
submitted in partial fulfillment of the requirements for
the Degree of Master of Science(Computer Science)

Acadia University
Spring Convocation 2022

This thesis by SIDDARDHA KAJA was defended successfully in an examination on December 08, 2021.

The examining committee for the thesis defense was:

Dr. Anna Kiefte, Chair

Dr. Haroon Malik, External Examiner

Dr. Martin Tango, Internal Examiner

Dr. Elhadi Shakshuki, Supervisor

Dr. Darcy Benoit, Head

Contents

List of Tables **xi**

List of Figures **xiii**

Abstract **xvii**

Definitions of Abbreviations and Symbols **xix**

Acknowledgements **xxi**

1 Introduction **1**

1.1 Problem Definition and Research Objective 2

1.2 Solution Overview 3

1.3 Scope 4

1.4 Contributions 4

1.5 Thesis Structure 4

2 Background Studies **7**

2.1 Internet Architecture 7

2.1.1 IP Address 9

2.1.2 Data Packets 9

Packet Flags 11

TCP NAK Option 12

2.1.3 Network Routing 13

2.2 Cloud Computing 13

2.3 Machine Learning . . . 15
2.3.1 Types of Machine Learning . . . 15
Supervised Learning . . . 15
Unsupervised Learning . . . 16
Reinforcement Learning . . . 16
2.4 Time Series Forecasting . . . 17
2.4.1 Stationarity . . . 18
2.4.2 Autocorrelation and Partial Autocorrelation . . . 18
Autocorrelation Function . . . 18
Partial Autocorrelation Function . . . 19
2.5 Time Series Models . . . 20
2.5.1 Persistence Algorithm . . . 20
2.5.2 Recurrent Neural Networks . . . 21
LSTM Architecture . . . 22
2.5.3 Deep Residual Networks . . . 24
N-Beats Architecture . . . 25

3 Design of Proposed Scheme and Approach 29
3.1 Overview . . . 29
3.2 Assumptions and Notations . . . 31
3.2.1 Assumptions . . . 31
3.2.2 Notations . . . 31
3.3 CACKS Design . . . 32
3.3.1 CACKS Modified Acknowledgement Packets . . . 33
3.3.2 CACKS Architecture . . . 34
3.4 Energy Aware Hybrid Scheduling Strategy . . . 44
3.4.1 Overview . . . 44
3.4.2 Meta-Scheduler . . . 44
3.4.3 Scheduling Model . . . 47
Variable Time Algorithm (VTA) . . . 47
Variable Frequency Algorithm (VFA) . . . 49

3.4.4 Transition Time . . . 52
3.4.5 Energy Model . . . 52
3.5 Machine Learning Approaches . . . 54
3.5.1 Data Collection . . . 54
3.5.2 Data Normalization . . . 57
Data Splitting . . . 57
3.5.3 Metrics Evaluation . . . 58
3.5.4 Machine Learning Approaches . . . 59
Persistence Model . . . 59
LSTM Model for Node Value Forecasting . . . 59
N - Beats Model for Node Value Forecasting . . . 60

4 Empirical Studies **63**
4.1 Data Collection . . . 63
4.2 Data Preparation . . . 66
4.3 Forecasting Model Development . . . 70
4.3.1 Objective . . . 70
4.3.2 Forecasting Models . . . 70
Persistence Model . . . 70
LSTM Model . . . 70
N-Beats Model . . . 72
4.4 Results . . . 76
4.4.1 Persistence Model Results . . . 76
4.4.2 LSTM Model Results . . . 78
First Split . . . 78
Second Split . . . 80
Third Split . . . 82
4.4.3 N-Beats Model Results . . . 84
4.4.4 Discussion . . . 86
4.5 Scheduling & Energy Models Simulation and Performance Evaluation . . . 87
4.5.1 CPU Utilization Pre-simulation . . . 87

4.5.2 Simulation Methodology 88
4.5.3 Simulation Configurations 89
4.5.4 Simulation Results 89

5 Conclusion and Future Work **91**
5.1 Conclusion 91
5.2 Future Work 92

Bibliography **93**

List of Tables

3.1 Description of Notations Used in Proposed Scheme 31
3.2 Data Packet Types and Data Packet 32
3.3 *BT-V* Matrix 50

4.1 ICMP Packets Route to Destination 64
4.2 LSTM Nested Cross-validation Split Information 72
4.3 N-Beats Hyperparameter Configurations 73
4.4 Nested Cross-validation Results of LSTM Model 86
4.5 Comparison of Three Forecasting Approaches 87

List of Figures

1.1 Traditional Routing Optimization Methods 2

2.1 Different Levels of Internet Service Providers 8
2.2 TCP Packet Anatomy [46] 10
2.3 Real World TCP Packet 11
2.4 Usual TCP Acknowledgement Packet Structure 12
2.5 Duty of Router in Network Routing [21] 13
2.6 Cloud Service Models [49] 14
2.7 Rolled RNN Chunk [52] 21
2.8 An Unrolled RNN [52] 21
2.9 LSTM Architecture [52] 22
2.10 A Residual Block [23] 25
2.11 N - Beats Basic Block [53] 26
2.12 N - Beats Architecture [53] 27

3.1 Modified ACK Packet 33
3.2 Modified NACK 34
3.3 Normal Network Control Flow 35
3.4 Cloud Acknowledgement Scheme Control Flow 36
3.5 Failed Intermediate Node Transmission Scenario 39
3.6 Scheduling Environment for the Cloud 45
3.7 Data and Control Flow of Meta-Scheduler 46
3.8 Energy Utilization Forecast [36] 53
3.9 Data Collection Process Flow Chart 56

3.10 Rolling Basis Cross Validation Technique [62] 58
3.11 Input Dimension Shape of LSTM 60
3.12 N-Beats Time Series in Consideration[53] 61

4.1 Geographical Location of the Selected Node 65
4.2 Last 100 Node Values in a Weekday 67
4.3 Last 100 Node Values in a Weekend 67
4.4 Last 100 Node Values in a Weekday 68
4.5 Last 100 Node Values in a Weekend 68
4.6 Node Values Time Series Decomposition 69
4.7 Observed Partial Autocorrelation of Time Series 69
4.8 LSTM Neural Network Architecture for Forecasting Packet Droppings . . . 71
4.9 N−Beats Neural Network Architecture for Forecasting Packet Droppings . . 75
4.10 Graphs of the Actuals vs Forecasts produced by Persistence Algorithm on a Weekday 77
4.11 Graphs of the Actuals vs Forecasts produced by Persistence Algorithm on a Weekend 77
4.12 Graphs of the Actuals vs Forecasts produced by First Split LSTM Model on a Weekday 79
4.13 Graphs of the Actuals vs Forecasts produced by First Split LSTM Model on a Weekend 79
4.14 Graphs of the Actuals vs Forecasts produced by Second Split LSTM Model on a Weekday 81
4.15 Graphs of the Actuals vs Forecasts produced by Second Split LSTM Model on a Weekend 81
4.16 Graphs of the Actuals vs Forecasts produced by Third Split LSTM Model on a Weekday 83
4.17 Graphs of the Actuals vs Forecasts produced by Third Split LSTM Model on a Weekend 83
4.18 Graphs of the Actuals vs Forecasts produced by N-Beats Algorithm on a Weekday 85

4.19 Graphs of the Actuals vs Forecasts produced by N-Beats Algorithm on a Weekend 85
4.20 CPU Utilization with Idle VM 88
4.21 CPU Utilization with Powering On and Off a VM 88
4.22 Time to Execute the Tasks Using VTA 90
4.23 Frequency of VMs Using VFA 90

Abstract

Every data packet has to pass through few intermediate nodes to reach its destination. Among other reasons, tremendous growth in internet devices encourages those intermediate nodes to drop the data packets. Optimizing the data packet route is an effective solution to deal with packet loss. Advanced machine learning approaches have been identified as a powerful support tool for routing optimization in node networks. Furthermore, as hardware infrastructure develops, the capabilities of cloud computing have expanded enormously. Improved connection, processing power, and memory units enable real-time machine learning. This thesis suggests and evaluates a unique technique for optimising the packet path by one hop for intermediate nodes as a backup called Cloud Acknowledgement Scheme. It offers information on the transmission trend and the tendencies of certain adjacent nodes or groups of neighbouring nodes in a network. We carried out a series of machine learning experiments and validated our idea using real-world node data.

Definitions of Abbreviations and Symbols

- ACK — Acknowledgement
- CACKS — Cloud ACKnowledgement Scheme
- IP — Internet Protocol
- IS-IS — Intermediate System to Intermediate System
- ISP — Internet Service Provider
- IXP — Internet Exchange Points
- LSTM — Long Short Term memory
- MAE — Mean Absolute Error
- NAK — Negative Acknowledgement
- NP — Nondeterministic Polynomial
- OSPF — Open Short Path First
- QoS — Quality of Service
- RNN — Recurrent Neural Network
- SMAPE — Symmetric Mean Absolute Percentage Error

- TCP — Transmission Control Protocol
- VFA — Variable Frequency Algorithm
- VM — Virtual Machine
- VTA — Variable Time Algorithm

Acknowledgements

I would first like to thank my supervisor, Professor Elhadi Shakshuki, who guided me in my research all the way from the beginning of my course. I am also grateful to Dr. Elhadi Shakshuki for his constant support, both professionally and personally during the COVID-19 hardships.

I would like to acknowledge my family and colleagues Sony Guntuka, Shashank Swarup, Hardik Manek, Deep Shah and my friends Mehak Lamba, Stephan Thomas, Nadine Wilson for their personal and emotional support.

I would also like to thank Dr. Daniel Silver and Dr. Andrew Mcintyre, for their valuable advises and knowledge at my Co-op work terms.

In addition, I would like to thank my parents Surendra Kaja and Madhavi Elisetty and my brother Suhas Kaja, who always supported me. Above all, I thank God almighty, for their showers of blessings throughout my research work.

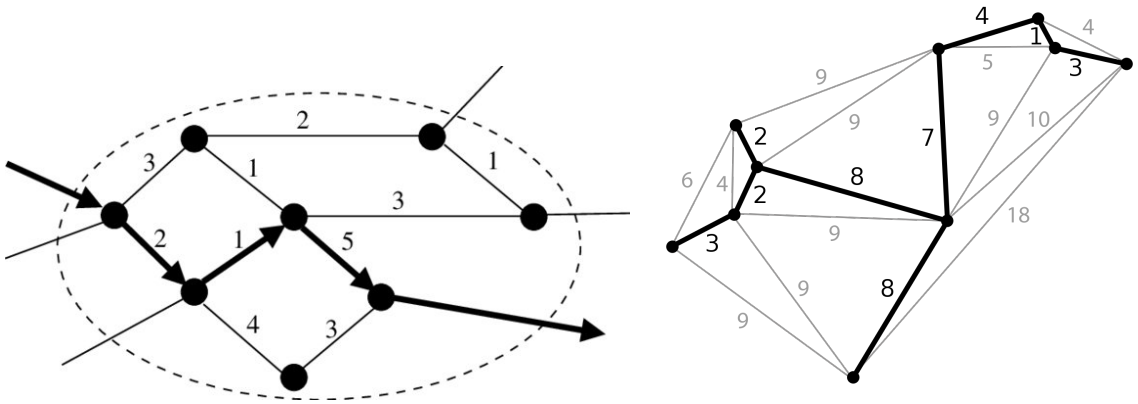
Chapter 1

Introduction

The amount of network devices, globally, connected to IP networks is projected to reach three times the global population in the near future [20]. This global trend in internet usage generates tremendous internet traffic across the planet. Among the traffic, Machine - Machine (M2M) applications such as smart meters, video surveillance, healthcare monitoring and IOT devices are the major contributors [59]. This means that every node in a network has to transmit its information efficiently and effectively. Powerful routing techniques enable networks to support the massive number of data transmissions.

Network routing is identifying a path that can communicate a data packet from a source node to a destination node. There are several different existing routing policies to determine the best route between nodes in a network. The common criteria for selecting the routing methods are Cost minimization, maximization of link utilization, QoS Provisioning [13] as shown in the Figure 1.1. Open Short Path First (OSPF) and Intermediate System to Intermediate System (IS-IS) are two great examples that enforce configuarable link weights to derive shortest paths [18][51]. Another type of well-known technique for routing optimization methods is spanning trees and Particle Swarm Optimization (PSO) [28]. Routing optimization techniques can be used for identifying the best route to obtain the best possible route between two nodes [21][60]. However, there are limitations for each of the existing routing optimization methods. They are having distance-vector metrics, 15-hop limitations, excessive routing traffic, and slow convergence are some of the drawbacks [6][22].

Therefore, routing optimization is an NP-hard problem that needs to be investigated



(a) Shortest Path Based on Link Weights [60] (b) A Minimum Spanning Tree [10]

Figure 1.1: Traditional Routing Optimization Methods

with a novel and innovative methodology. Taking this into consideration, rapidly developing technologies like cloud computing and machine learning have the potential to boost routing optimization through new approaches [19]. Real-time machine learning with the help of the cloud provides new potential for enhancing routing optimization [50][39].

1.1 Problem Definition and Research Objective

The purpose of this research is to design an approach that assists a node in identifying the best node among all the neighboring nodes to retransmit a data packet whenever a node encounters a packet drop scenario. The best node refers to the situation when there is a high likeliness in the next packet transmission through that particular node will be a successful transmission. A survey on machine learning techniques for routing optimization suggests that there are successful methodologies that help in route optimization [43]. However, all these techniques compel all the nodes in a network to implement their methodologies. Thus the objective is to provide machine learning-driven suggestions to the nodes in a network as a backup option. More complications are expected to develop machine learning models to identify and predict the packet drop patterns. Selfish or malicious nodes in a network usually follow a certain packet dropping pattern in a specific period. It will be challenging to forecast the packet droppings without prior observations.

1.2 Solution Overview

The researcher proposes a novel approach using a cloud acknowledgment scheme (CACKS) and machine learning techniques. CACKS exploits acknowledgment packets that are generated by the nodes in a network and offers backup services to nodes affected by failed transmissions due to packet droppings in the network. In CACKS, the cloud collects all the acknowledgment packets produced in a network and uses them to extract knowledge about a particular network. The packets provide information on the active nodes that can communicate and also on all other data-packet transmissions across the network as well. All the data is collected by the cloud and used to monitor the network. This phase hybridizes a network's environment by reintroducing centralization in a network while preserving the network's decentralized character.

The collected node values are then portrayed as time-series data that are later used to observe the trends and seasonalities of the network nodes. Advances in machine learning techniques have allowed for real-time predictions of trends [35]. In this research, to uncomplicate the data and model, data was collected as univariate time series. Therefore univariate time series data is very simple and less burdensome on the cloud to forecast the tendency of a node. LSTM and N-Beats models are used to develop a predictive model based on previous transmission outcomes. Both LSTM and N-Beats models are compared with each other as well as with a persistence model for accuracy. Persistence algorithm, the naïve forecast method is the most common baseline method that utilizes persistence algorithm for time series data. This model forecasts the values based on previous transmission outcomes. LSTM models are deep recurrent neural networks that can provide the forecast of a node tendency based on a long sequence of previous transmission outcomes. LSTM can perfectly handle long-term dependencies and can capture seasonalities in the packet droppings of the node transmission data. N-Beats models are deep and fully connected residual neural network that deals with univariate time series data which is perfect for this research. Furthermore, N-Beats achieved the best univariate time series forecasting model in the M4 competition.

The researcher expected that it is considerably more challenging to build and test a platform without a robust packet-level network simulation tool that can integrate machine learning methods during this research. For this reason, the researcher aims to prove that node

packet droppings can be forecasted and pseudocode is developed to achieve the objective by utilizing cloud services.

1.3 Scope

Seven days of data is collected by observing a node's transmissions. Continuous ICMP packets with 64 bytes are transmitted to a node every sixty seconds. Successfully transmitted data packet's response is collected and organized along with the sent data packets. For every successful packet transmission, a reward of +1 is given and a penalty of -1 is given to this node whenever this node drops a packet. Data is collected by taking the aim of this research into consideration that is to be able to forecast the packet droppings based on previous transmission outcomes.

1.4 Contributions

The principal contributions of this thesis are:

- A novel cloud-node communication model is introduced in order to allow the cloud to monitor the packet transmissions across the network.
- Three machine learning models are built, trained, and tested for the cloud to assist nodes that need to help in data packet retransmission whenever they experience packet droppings.
- A dataset is developed with the trasnmission history of a selected node by observing a particular node for few days in order to train the machine learning models with a node's packet dropping background.

1.5 Thesis Structure

The rest of this thesis is structured as follows. Chapter 2 provides a review of background information. The proposed scheme and approach are designed in detail in Chapter 3. In

Chapter 4, empirical investigations provide the findings of this research. Finally, in Chapter 5, the conclusion is provided.

Chapter 2

Background Studies

2.1 Internet Architecture

The architecture of the internet is always dynamic because of the endless transformation of technologies, topologies, devices, and services. Considering the scale of the internet, it is very challenging to describe the structure of the internet [30]. Streamline internet architecture is explained as a distributed system made up of multiple smaller networks owned by Internet Service Providers (ISP), universities, governments, and other organizations that are linked together with peering agreements.

Broadly, the internet can be outlined into three levels as shown in the Figure 2.1. They are:

- Backbone ISPs

  The internet backbone is the collection of multiple smaller networks. It consists of access links that bring the smaller network traffic to high bandwidth routers that transmit the traffic from its source node to the destination over the best available path. Usually, giant network carriers called Tier 1 ISPs like AT&T, Cogent Communications, Deutsche Telekom, NTT Communications, Tata Communications, Telia Carrier and Verizon are some of the major backbone providers. Collectively these networks create a massive worldwide network that has the entire routing table. Backbone ISPs are connected to each other at Internet Exchange Points (IXP) also called peering

points. It is the location of high-speed routers and switches that move traffic among backbone ISPs. It is not owned by any of the Tier 1 ISP instead funded by them but do not charge each other for high-speed transmissions among the backbone ISPs. This interdependence is called settlement-free peering. Usually, backbone ISPs equip the available fastest routers with upto 100 gbps trunk speed. Companies like Cisco, Extreme, Huawei, Juniper and Nokia offer these high speed routers and switches [30].

- Regional ISPs

  Backbone ISPs provide access to their high-speed high-bandwidth routers and switches to smaller Tier 2 and Tier 3 ISPs. Tier 3 ISPs provide internet connections to the clients like individuals and businesses. However, Tier 3 ISPs have no connection to the larger backbone networks. Hence the Tier 2 ISPs provide Tier 3 ISPs access to their small regional networks. Simultaneously, Tier 2 ISPs buy access to the expensive global backbone networks.

- Clients

  Clients are the consumers of the internet. Businesses and individual homes subscribe to the Tier 3 ISPs for internet connection.

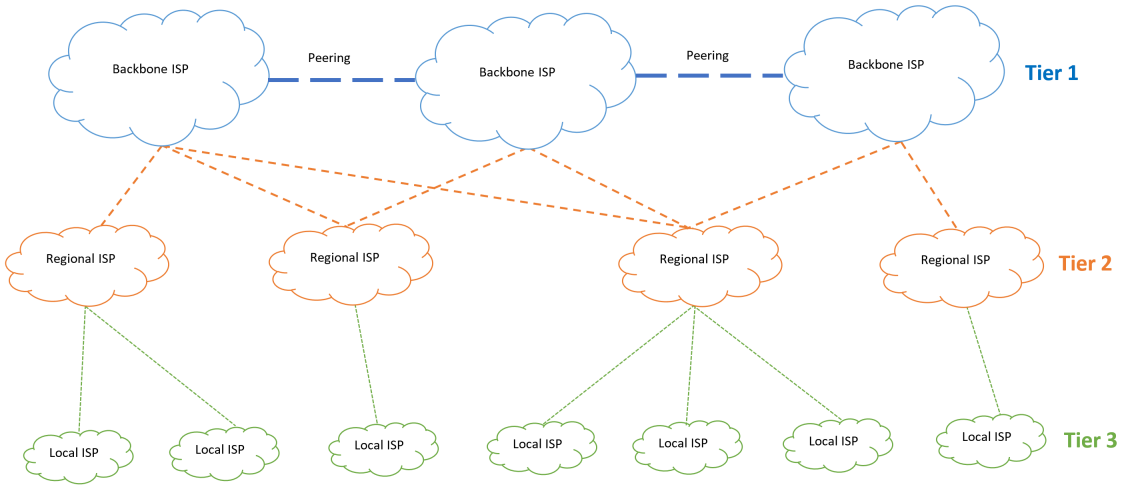


Figure 2.1: Different Levels of Internet Service Providers

2.1.1 IP Address

For the routers or switches to transmit data packets across the networks, addresses of the devices attached to the network are necessary. Each device connected to a network has its unique address called Internet Protocol (IP) address. The IP address might be unique but they are not always permanent [9]. Generally, there are two different types of IP addresses. They are public IP addresses and private IP addresses. Every computer, smartphones, IOT devices, Bluetooth speakers have their private IP addresses. However, within public IP addresses, there are two types of are IP addresses that need to be discussed [38]. They are:

- Dynamic IP addresses

  ISP often purchases a large pool of IP addresses from the backbone or regional ISPs and assigns them to their customers. However, they are regularly re-assigned to other customers or put back into the un-assigned pool of IP addresses. Constantly changing IP addresses is a way to secure devices against hackers.

- Static IP addresses

  On the other hand, static IP addresses are in contrast to dynamic IP addresses. Some devices like servers, powerful switches, and high-bandwidth routers need to be fixed at a location. A constant IP address needs to be assigned to these kinds of devices. Hence, static IP addresses that most likely never change over a period of time are assigned to them.

2.1.2 Data Packets

The network transmissions happen in the form of data packets. A format and a medium of transmission are essential for a node to communicate with another. Data packets act as a means to carry data or messages. They are the smaller chunks of a larger message [5]. Data packets not only hold the data but also carry a certain type of metadata for addressing, error correction, type of the data packet, routing information and organizing data packets.

Figures 2.2 and 2.3 displays the architecture of a Transmission Control Protocol (TCP) packet architecture through a basic figure of TCP anatomy and real-world TCP packet details.

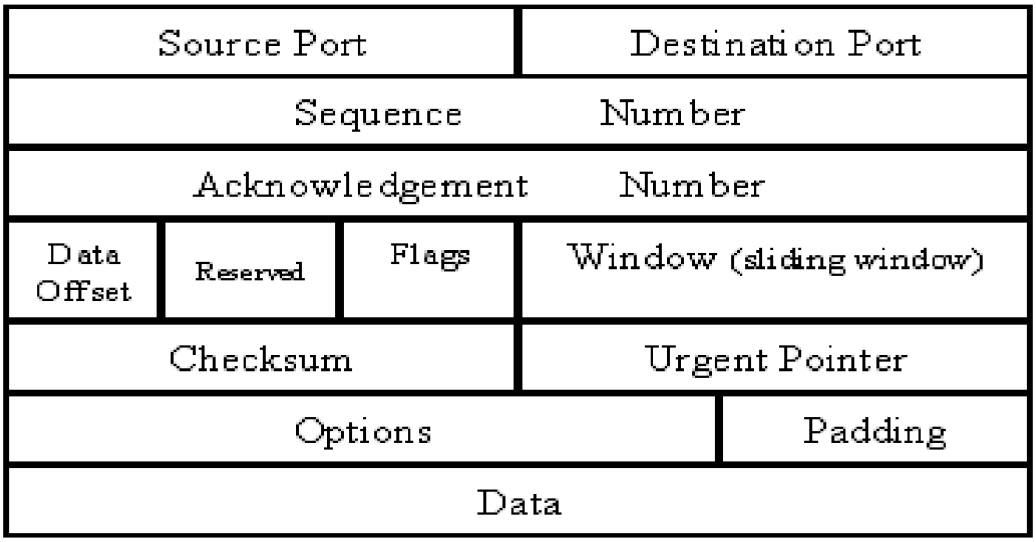


Figure 2.2: TCP Packet Anatomy [46]

Figure 2.2 shows the constituents of a TCP packet. Along with the data, it contains the metadata of the TCP packet like [46]:

- Source port and destination port to address the end points of the connection.
- Sequence number of the first byte of the data in TCP data packet
- Acknowledgement number is the sequence number of the next data packet
- Data offset is the length of the header
- Reserved field the empty field for the future use
- Flags are for indicating the type of the data packet
- Window size is the buffer space of the sender's receiving Window
- Checksum is used for error correction of the data packet
- Urgent pointer field provides the information of the byte that needs to be addressed urgently
- Options field specifies the various TCP options like Maximum Segment Size (MSS), Window Scaling, Selective Acknowledgements, Timestamps and No Option (nop)

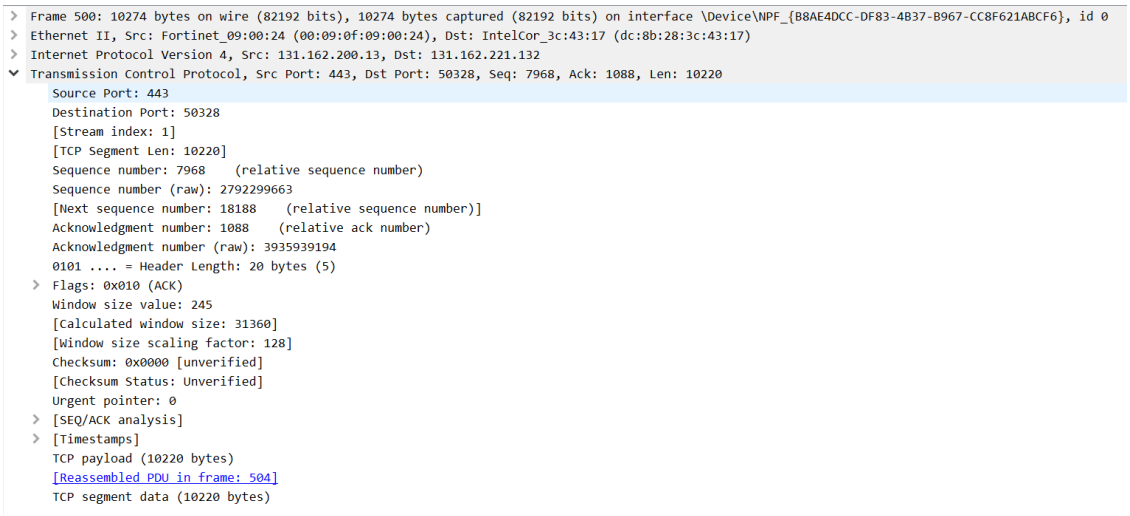


Figure 2.3: Real World TCP Packet

Packet Flags

There are different kinds of TCP packets. To indicate the type of the TCP packet, packet flags are used. Packet flags also provide additional information like troubleshooting purposes or to manage the operations of a particular connection along with the state of the connection [12]. Each TCP flag size is about 1 bit. There are 7 different types of TCP packet flags. They are:

- ACK - Acknowledgement flag is used to indicate the successful receipt of a packet.
- FIN - Finish flag is used to let the receiver know that there are no more data packets to send and indicates the end of transmission
- URG - Urgent flag informs the destination to process urgent packet before processing the other packets
- PSH - Push flag informs the transport layer to transmit push packets from application layers to network layer without any delay
- RST - Reset flag is used to terminate the connection because of unexpected transmission to another host that is not expecting the TCP packet.

- ECE - ECN - Echo (Explicit Congestion Notification - Echo) denotes if the TCP peer has ECN option.
- CWR - Congestion Window Reduced flag is used if the sender receives a ECE flag packet.

Figure 2.4 displays the actual acknowledgement packet with packet flag set to ACK. Acknowledgment packets do not hold any payload. However, it contains the metadata of a TCP packet similar to a regular TCP packet that carries data.

```
Transmission Control Protocol, Src Port: 443, Dst Port: 54347, Seq: 1, Ack: 40, Len: 0
  Source Port: 443
  Destination Port: 54347
  [Stream index: 2]
  [TCP Segment Len: 0]
  Sequence number: 1    (relative sequence number)
  Sequence number (raw): 2340598118
  [Next sequence number: 1    (relative sequence number)]
  Acknowledgment number: 40    (relative ack number)
  Acknowledgment number (raw): 1135786310
  0101 .... = Header Length: 20 bytes (5)
> Flags: 0x010 (ACK)
  Window size value: 669
  [Calculated window size: 669]
  [Window size scaling factor: -1 (unknown)]
  Checksum: 0x1e0a [unverified]
  [Checksum Status: Unverified]
  Urgent pointer: 0
v [SEQ/ACK analysis]
    [This is an ACK to the segment in frame: 35]
    [The RTT to ACK the segment was: 0.038145000 seconds]
> [Timestamps]
```

Figure 2.4: Usual TCP Acknowledgement Packet Structure

TCP NAK Option

Among other options, Request For Comments (RFC) 1106 suggests the use of Negative Acknowledgement packets (NAKs) [27]. This option allows the destination node to inform the sender that a data packet is not received or needs retransmission. This option is useful if the nodes in the path experience any periodic errors like packet droppings, packet loss, or noisy links. Following the information provided by this option is voluntary. It does not have any effect on the TCP transmissions if ignored.

2.1.3 Network Routing

A data packet requires traveling across one or more networks to reach its destination. Network routing is the procedure to identify the best path from the sender node to the destination node. For the internet, routing decisions are made by routers in the network. Figure 2.5 illustrates a decision making scenario by a router. The router connected to computer A needs to decide the best route to reach computer B. The path where Network 2 and 4 seems short but it has high latency. However, Networks 1, 3, and 5 have low latency but it is a longer route. The router needs to decide the path for the data packets.

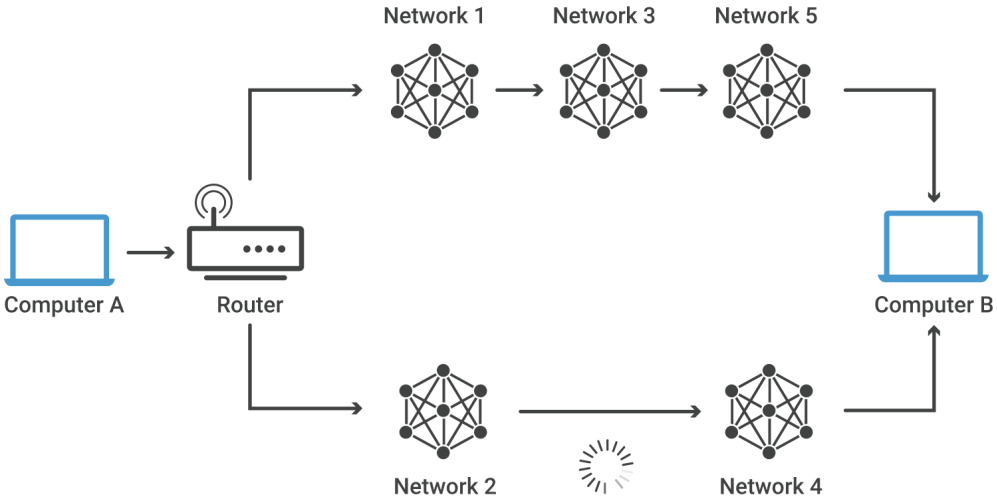


Figure 2.5: Duty of Router in Network Routing [21]

Routers usually refer to the internal routing tables to determine the route to a destination node. The network layer in the devices is responsible for the routing process to deliver data packets by choosing the optimal path.

2.2 Cloud Computing

Cloud computing is the on-demand delivery of computing resources via the internet. The services include applications, physical &virtual servers, data storage, development tools, networking capabilities and processing power. These services can be hosted by remote servers as well as on-premise servers.

Virtualization is the key technology that allows the cloud environment to share the physical instance of a resource or an application among multiple customers or organizations [3]. It allows multiple customers or organizations to share the same application from different locations. Similarly, single physical hardware is emulated into multiple virtual environments called virtual machines (VM). A software called hypervisor abstracts the underlying machine's resources like processing power, storage and cloud-based applications and lets them be allocated into centralized pools that are available for cloud service deployment as virtual machines [7].

There are different models of cloud services. They are illustrated in the Figure 2.6 as:

- IaaS - Infrastructure as a Service refers to the provision of fundamental computing resources like essential compute, storage and networking [47].
- PaaS - Platform as a Service includes services like development tools, database management and business analytics. Moreover, operating systems can also be deployed along with all the services available with IaaS [48].
- SaaS - Software as a Service is the delivery of software application as a service over the internet [49].

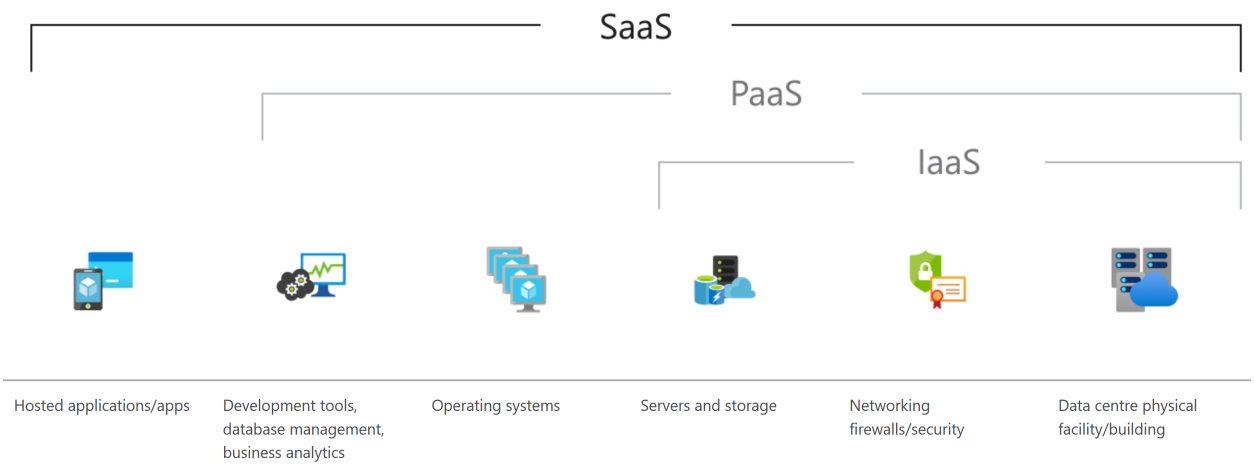


Figure 2.6: Cloud Service Models [49]

2.3 Machine Learning

According to Arthur Samuel who coined the term Machine Learning (ML), it is defined as "Field of Study that gives the computers the capabilities to learn without being explicitly programmed" [11]. ML is a branch of Artificial Intelligence (AI) and computer science that makes use of data and algorithms to imitate human learning capabilities. Machine learning algorithms can extract patterns in data and learn from them, to make their predictions. Therefore, machines are capable of solving complex problems and take actions with little or no human intervention by automating the process [1]. The basic assumption of ML is to build and train models that can receive input data and use statistical analysis to obtain the output. In 1997, Tom Mitchell defined ML mathematically and relationally as,

> a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [2].

2.3.1 Types of Machine Learning

Machine Learning can be classified into 3 types of algorithms. They are detailed in the following subsections.

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Supervised Learning

Supervised learning algorithms are mathematical machine learning models that train on a collection of data points that includes both inputs and corresponding outputs. Dataset D, for example, is a collection of data that includes a as inputs and b as output labels. Dataset D is divided into two subsets: a training set and a testing set. An ML algorithm learns the hypothesis function $h(a)$, which maps the input space A to the output space B. The learnt

mapping function *h(a)* approximates the real mapping function *f(a)* between A and B, as shown in the equation 2.1. The ML algorithm learns a hypothesis *h(a)* from the training set, which is subsequently tested on the testing set to yield an error e. The error e quantifies how successfully the ML algorithm generalizes inputs outside of D. The outputs b for a classification issue are referred to as the class, and the challenge of determining the class for a given a is referred to as classification.

$$f(x) = X \rightarrow Y \tag{2.1}$$

Another notable concept in supervised learning is regression. The job of estimating a mapping function *f(a)* from input variables a to a continuous output variable b is known as regression predictive modelling. Often the output variable y is a real-value and these are often quantities like sizes and amounts [16].

Unsupervised Learning

Unsupervised learning is a sort of machine learning technique that is used to derive conclusions from datasets that contain input data but no labeled answers. We frequently do not have clear labels for datasets, and even if we did, we would only have labels for a very tiny percentage of the cases. This makes supervised learning harder for us to apply. When the class b of the input data a is not provided, machine learning is referred to as unsupervised. The learning algorithm splits the training set's samples into groups in an attempt to group comparable cases. These groups are known as clusters, and the process is known as clustering. Although there are several clustering methods, approaches such as K-Means or Self-Organizing Maps (SOM) are favored because they are simpler to visualize [40].

Reinforcement Learning

Reinforcement learning is the process of teaching machine learning models to make a series of judgments. In an uncertain, possibly complicated environment, the agent learns to attain a goal. A reinforcement model is presented in a game-like setting in reinforcement learning. The computer uses trial and error to find a solution to the problem. To persuade the computer to do what the programmer desires, the model is either rewarded or punished for

the acts it does. Its strategy is to improve the overall return. Despite the fact that the programmers choose the incentive policy, they provide no clues or ideas for how to solve the game to the model. It is up to the model to discover how to do the job in order to maximize the reward, beginning with completely random trials and progressing to complex tactics and superhuman abilities. Reinforcement learning is presently the most effective approach to hint at machine creativity by harnessing the power of search and numerous trials. Unlike humans, artificial intelligence can learn from thousands of simultaneous gameplays if a reinforcement learning algorithm is performed on a strong enough computer infrastructure [54].

2.4 Time Series Forecasting

Time series forecasting is an important part of machine learning. It is the crude concept behind all machine learning based forecasting techniques [15]. Time series forecasting uses the date and time elements in the data for better forecasting. Time series analysis is a significant approach to involve to understand the different components of the time series data; these components are often characterized as trend, seasonality, and cycle. Each of these components portrays insights to various aspects of the data that can contribute to accuracy in time series forecasting.

- **Trend** - Trend is defined as long term increase or decrease in the data. It can be linear or nonlinear;
- **Seasonality** - Variations that repeat over a specific period such as a day, week, month, season, etc.;
- **Cycle** - A cyclic pattern exists when data exhibit rises and falls that are not of fixed period.

Traditionally, most time series data can be classified into two types regular and irregular time series. If data is collected after every specific time period then it is considered as regular time series, otherwise it is irregular time series data. Most of the stock price forecasting models use data sets where points (values at a specified date and time) are recorded after every second. Natural disaster time series in the form of volcano eruptions, earthquakes

and floods data are recorded irregularly. Similarly, time series can also be classified into univariate and multivariate time series [24].

- **Univariate time series** - Univariate time series data consists of data with just the time step and its data point. It does not hold any other variables that affect the target (predicted) data point. Forecasting models for univariate time series consider only previous values of the variable. These models are also referred to as 'auto-regressive.
- **Multivariate time series** - Multivariate time series incorporates a variety of features into the dataset. These features affect the current or future of the data point in some way. Multivariate time series forecasting considers previous values of the variable as well as other features. For example, humidity, air pressure and precipitation, etc, are some of the features that play a major role in weather forecasting.

2.4.1 Stationarity

Stationarity is the property of a time series dataset. It means that the mean and variance ceases to vary over time. Generally, most of the time series techniques consider that the data is stationary. However, most time series datasets are non-stationary, which means they have fluctuating components. Trend, Seasonality, and Cycles or a combination of the three components can be observed in non-stationary time series data points [34]. The data points in the stationary time series do not possess a temporal structure.

2.4.2 Autocorrelation and Partial Autocorrelation

Time series analysis and forecasting make extensive use of AutoCorrelation and partial autocorrelation plots.

Autocorrelation Function

When it comes to AutoCorrelation Function (ACF), the series must be weakly stationary for an ACF to make sense. This indicates that the autocorrelation for any given lag is the same regardless of time.

If a series x_t meets the following characteristics, it is considered to be stationary:

- The mean $E(\mathrm{x_t})$ for all t is the same.
- The variance of x_t for all t is the same.
- The covariance between x_t and x_{t-1} for all t is the same.

At time t, let the value of the time series be denoted by x_t. The correction between x_t and x_{t-h} for $h = 1, 2, 3, etc.$ of a given time series is provided by the ACF. In theory, the autocorrelation between x_t and x_{t-h} is presented in the equation 2.2

$$\frac{Covariance(x_t, x_{t-h})}{Std.Dev.(x_t)Std.Dev.(x_{t-h})} = \frac{Covariance(x_t, x_{t-h})}{Variance(x_t)} \tag{2.2}$$

Because the standard deviation of a stationary series is constant, the denominator in the second expression happens. The last characteristic of a weak stationary series states that the theoretical value of autocorrelation for a specific lag is the same across the whole series. A stationary series has the same structure ahead as it does backward, which is an intriguing characteristic. ACF patterns may be seen in many stationary series. However, the vast majority of series seen in practice are not stationary. A persistent rising trend, for example, violates the condition that the mean is the same for all t. Distinct seasonal patterns are also in violation of this criterion [54].

Partial Autocorrelation Function

A partial correlation is a conditional correlation in general. It is the relationship between two variables under the premise that we know and consider the values of another set of variables. Consider the following scenario: y is the response variable, while A, B, and C are predictor variables. The partial correlation between y and C is the correlation obtained by taking into consideration how both y and C are connected to A and B.

This partial connection might be discovered in regression by comparing the residuals from two separate regressions. First regression is the regression in which we predict y from A and B. Second regression is the regression in which we predict C from A and B. Essentially, we correlate the parts of y and C that A and B do not predict. Equation 2.3 defines the partial autocorrelation [54].

$$\frac{Covariance(y, C|A, B)}{\sqrt{Variance(y|A, B)Variance(C|A, B)}} \tag{2.3}$$

2.5 Time Series Models

The basic concept of time series forecasting is to predict the timestep y assuming that it has a relationship with its previous timesteps. There are many ways to model a time series. Among them, the following are three types of solutions for our time series forecasting problem.

- Persistence Algorithm
- Recurrent Neural Networks - Long Short Term Memory architecture
- Residual Neural Networks - N-Beats architecture

2.5.1 Persistence Algorithm

The persistence model is frequently used as a reference for determining the skill factor. It is useful to determine whether a prediction model outperforms any simple reference model, such as the persistence model. The persistence method is arguably the easiest approach to generate a prediction. A persistence theory assumes that the future value of a time series is determined on the premise that nothing changes between the present time and the prediction time. For any time series problem, the persistence model predicts that the timestep value at time t equals the timestep value at time $t+1$ as shown in the Equation 2.4. For this reason, the persistence algorithm is also known as the naive predictor [55].

$$timestep_{\mathrm{t}} = timestep_{\mathrm{t+1}} \tag{2.4}$$

The model's accuracy declines as the forecast time horizon increases, and this model is typically insufficient for time horizons longer than one hour. The persistence model is used as a reference in this context to measure the reliability of the other two models.

2.5.2 Recurrent Neural Networks

Traditional neural networks have shortcomings, which recurrent neural networks solve. Traditional neural networks were unable to integrate reasoning about earlier events in the timeline to inform future outcomes. Recurrent neural networks are networks that contain loops that allow information to be retained.

Figure 2.7 displays the chunk of neural network. Here, A takes in some input x_t and returns a value h_t. A loop allows data to be transferred from one network phase to the next.

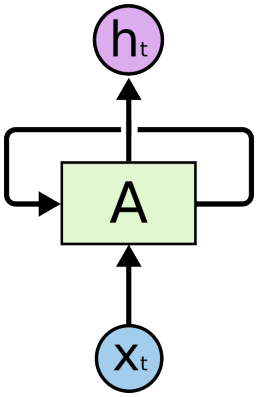


Figure 2.7: Rolled RNN Chunk [52]

Because of these loops, recurrent neural networks appear strange. However, they are not all that different from a conventional neural network. A recurrent neural network may be thought of as many clones of the same network, each delivering a message to the next in line. When the loop is unrolled as shown in the Figure 2.8, the chain-like structure of recurrent neural networks demonstrates that they are closely linked to sequences and lists. They are by far the most natural neural network architecture to apply for this type of data.

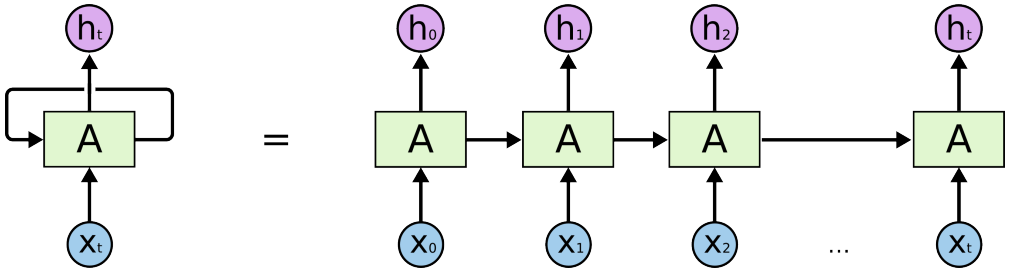


Figure 2.8: An Unrolled RNN [52]

LSTM Architecture

Long Short Term Memory networks, abbreviated as LSTMs, are a kind of RNN capable of learning long-term dependencies. Hochreiter and Schmidhuber (1997) introduced them, and numerous others developed and popularised them in subsequent work. They perform remarkably well on a wide range of problems and are now extensively applied.

LSTMs are deliberately designed to avoid the problem of long-term reliance. Remembering knowledge over extended periods is essentially their default behavior. All recurrent neural networks take the form of a chain of repeating neural network modules with a single tanh layer. This recurring module in a typical RNN will have a pretty straightforward structure. Although LSTMs have a chain-like structure, the repeating module has a different structure. Instead of a single neural network layer, there are four, each interacting distinctively as shown in the Figure 2.9.

The horizontal line going across the top of the Figure 2.9 is the key to LSTMs. The cell state is similar to a conveyor belt. It follows the whole chain, with only a few small linear interactions. It is relatively simple for information to just travel over it unaltered.

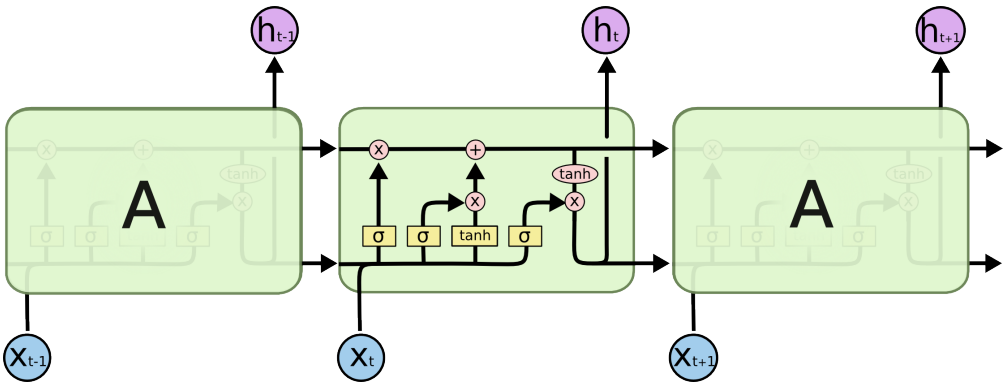


Figure 2.9: LSTM Architecture [52]

The LSTM may delete or add information to the cell state, which is carefully regulated by structures known as gates. They are a method of allowing information to pass through if desired. They consist of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer produces values between 0 and 1, indicating how much of each component should be allowed through. A value of zero indicates that nothing is allowed to pass through. A value of one indicates that everything is allowed to pass through. An LSTM has three of

these gates to maintain and regulate the cell state. They are a forget gate, an input gate, and an output gate.

If a basic LSTM cell is observed, the first step of the LSTM is to select what information needs to be discarded from the cell state. This choice is determined through a sigmoid layer known as the forget gate layer. It examines h_{t-1} and x_t and returns a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 denotes that the information should be kept in its entirety. A 0 signifies entirely discarding the passed information. The forget gate is defined in the Equation 2.5. where w_f is the weight, h_{t-1} is the output from the previous time step, x_t is the current input and b_f is the bias.

$$f_{\mathrm{t}} = \sigma(w_{\mathrm{f}} * [h_{\mathrm{t\text{-}1}}, x_{\mathrm{t}}] + b_{\mathrm{f}}) \tag{2.5}$$

The following step is to decide what additional information we will store in the cell state. There are two components to this. First, a sigmoid layer known as the "input gate layer" with a sigmoid function i_t determines which values will be updated as defined in the Equation 2.6. Following that, a tanh layer generates a vector of new cell values as defined in the Equation 2.7, $\widetilde{C}_t$, that might be added to the state. In the following phase, we will combine these two to generate a state update.

$$\widetilde{C}_{\mathrm{t}} = tanh(w_{\mathrm{C}} * [h_{\mathrm{t\text{-}1}}, x_{\mathrm{t}}] + b_{\mathrm{C}}) \tag{2.6}$$

$$i_{\mathrm{t}} = \sigma(w_{\mathrm{i}} * [h_{\mathrm{t\text{-}1}}, x_{\mathrm{t}}] + b_{\mathrm{i}}) \tag{2.7}$$

The previous cell state, C_{t-1}, must be replaced with the new cell state, C_t. The preceding phases have already determined what we should multiply the old state by f_t, forgetting what we agreed to forget previously. Then $i_t * \widetilde{C}_t$ is pushed to the new cell value, scaled by the amount by which we opted to change each state value. The new cell state is formed by the Equation 2.8.

$$C_{\mathrm{t}} = C_{\mathrm{t\text{-}1}} * f_{\mathrm{t}} + t_{\mathrm{t}} * \widetilde{C}_{\mathrm{t}} \tag{2.8}$$

In the final stage, we must decide what we will produce as output. This output will also

be dependent on our cell status, but it will be adjusted. First, we run a sigmoid layer to determine which bits of the cell state will be output as defined in the Equation 2.9.

$$o_{\mathrm{t}} = \sigma(w_{\mathrm{o}} * [h_{\mathrm{t\text{-}1}}, x_{\mathrm{t}}] + b_{\mathrm{o}}) \tag{2.9}$$

To obtain the output value $\mathrm{h_t}$, the new state $\mathrm{C_t}$ is passed through a tanh layer and multiplied by the output gate, as indicated in the Equation 2.10.

$$h_{\mathrm{t}} = o_{\mathrm{t}} * tanh(C_{\mathrm{t}}) \tag{2.10}$$

2.5.3 Deep Residual Networks

According to research on the prediction accuracy of neural networks, increasing the depth of a network frequently results in increased performance. However, merely stacking a high number of layers in a network is not a reliable way to improve its speed [32]. Increasing the depth of a network not only increases the computing cost of training it but can also make training more challenging. The basic concept of ResNet is to provide a so-called identity shortcut connection that bypasses one or more levels, as seen in the Figure 2.10 below. The formulation of the building block is defined in the Equation 2.11. where x and y are the input and output vectors of the respective layers under consideration. The residual mapping to be learned is represented by the function *F(x, W_i)*

$$y = F(x, W_{\mathrm{i}}) + x \tag{2.11}$$

When the Equation 2.11 is applied to the example in the Figure 2.10 that has two layers, then $F = W_2\sigma(W_1x)$, where σ denotes ReLu and the biases are ignored for less complications. A shortcut connection and element-wise addition are used to execute the operation $F + x$. The shortcut connections in Equation 2.11 add no more parameters or complexity to the computation. After the addition, a second nonlinearity is used.

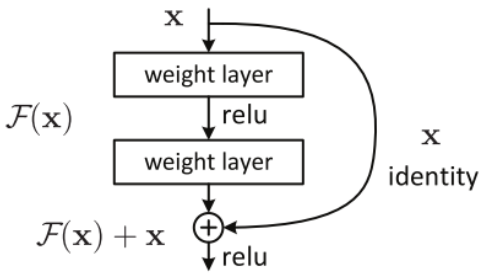


Figure 2.10: A Residual Block [23]

N-Beats Architecture

N-Beats design approach is founded on two fundamental concepts. To begin, the foundation architecture should be simple and general, yet expressive. Second, the design should not rely on feature engineering or input scaling that is time-series specific. The suggested basic building block of N-Beats has a fork design and is illustrated in Figure 2.11. For example, take the *l-th* block, which accepts its corresponding input x_l and produces two vectors $\hat{x}_l$ and $\hat{y}_l$. The first block in the model's x_1 is the overall model input a historical lookback window of a particular duration ending with the most recently observed observation. The length of the input window is chosen to be a multiple of the forecast horizon H, and typical values of x in our configuration vary from 2H to 7H. The remaining blocks, inputs x_l are the residual outputs of the preceding blocks. Each block produces two outputs: $\hat{y}_1$, the block's forward forecast of length H, and $\hat{x}_1$, the block's best estimate of x, sometimes known as the backcast, given the functional space restrictions that the block can employ to approximate signals. The fundamental block is divided into two sections on the inside. The first portion is a fully linked network that generates the forward θ_l^f and backward θ_l^b expansion coefficient predictors. The backward g_l^b and forward g_l^f basis layers receive the corresponding forward θ_l^f and backward θ_l^b expansion coefficients, project them internally on the set of basis functions, and create the backcast $\hat{x}_l$ and forecast outputs $\hat{y}_l$. The following equations describe the functioning of the first section of the *lth* block as formulated in the Equation 2.12 [53].

$$h_{1,1} = FC(x_1), h_{1,2} = FC_{1,2}(h_{1,1}), h_{1,3} = FC_{1,3}(h_{1,2}), h_{1,4} = FC_{1,4}(h_{1,3}). \tag{2.12}$$

$$\theta_l^b = Linear_l^b(h_{l,4}), \theta_l^f = Linear_l^f(h_{l,4}) \tag{2.13}$$

The LINEAR layer is a fully connected linear projection layer. This section of the architecture is responsible for predicting the forward expansion coefficients θ_l^f, with the ultimate objective of maximizing the accuracy of the partial prediction $\hat{y}_l$ by appropriately mixing the basis vectors given by g_l^f. Furthermore, this section of the network predicts the backward expansion coefficients θ_l^b utilized by g_l^b to provide an estimate of x_l, with the ultimate objective of assisting downstream blocks by eliminating components of their input that are not useful for forecasting. The network's second section transfers expansion coefficients θ_l^f and θ_l^b to outputs through basis layers $\hat{y}_l = g_l^f(\theta_l^f)$ and $\hat{x}_l = g_l^b(\theta_l^b)$. The following equation 2.14 illustrate how it works:

$$\hat{y}_l = \sum_{i=1}^{dim(\theta_l^f)} \theta_{l,i}^f v_l^f, \hat{x}_l = \sum_{i=1}^{dim(\theta_l^b)} \theta_{l,i}^b v_l^b. \tag{2.14}$$

$\theta_{l,i}^f$ is the i$^{\text{th}}$ element of θ_l^f, while v_i^f and v_i^b are the forecast and backcast basis vectors. The role of g_l^b and g_l^f is to generate sufficiently rich sets $(v_i^f)_{i=1}^{dim(\theta_l^f)}$ and $(v_i^b)_{i=1}^{dim(\theta_l^b)}$ such that their respective outputs can be appropriately represented by changing expansion coefficients θ_l^f and θ_l^b.

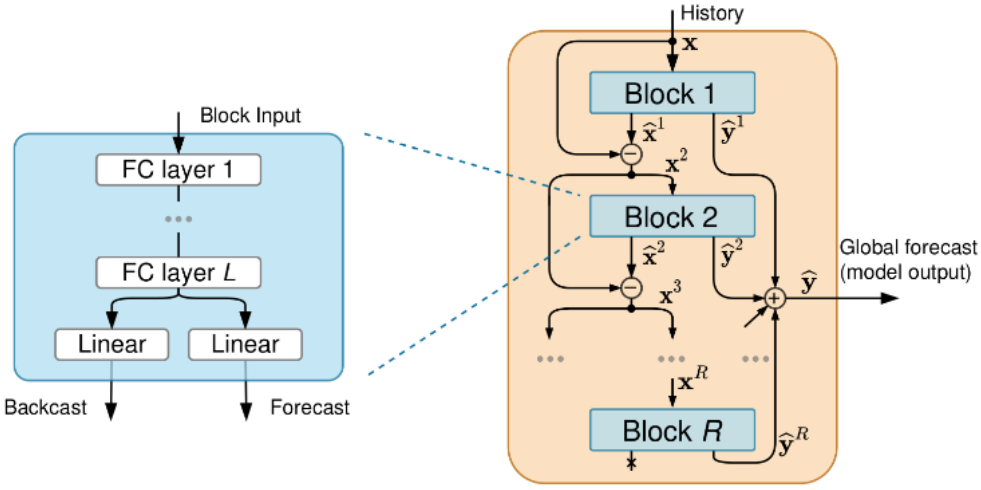


Figure 2.11: N - Beats Basic Block [53]

Blocks are arranged into stacks so that the current block's backcast is subtracted from its input and given as input to the next block, and the forecast vectors from each block are combined to create the overall stack forecast as shown in the figure 2.12. The stacks are linked together in a pipeline, with the backcast output of one stack serving as the input for the next. The overall model forecast is then produced by adding the forecasts from all stacks. The N-BEATS architecture has been shown to provide excellent performance on a variety of commonly used time series forecasting datasets [53].

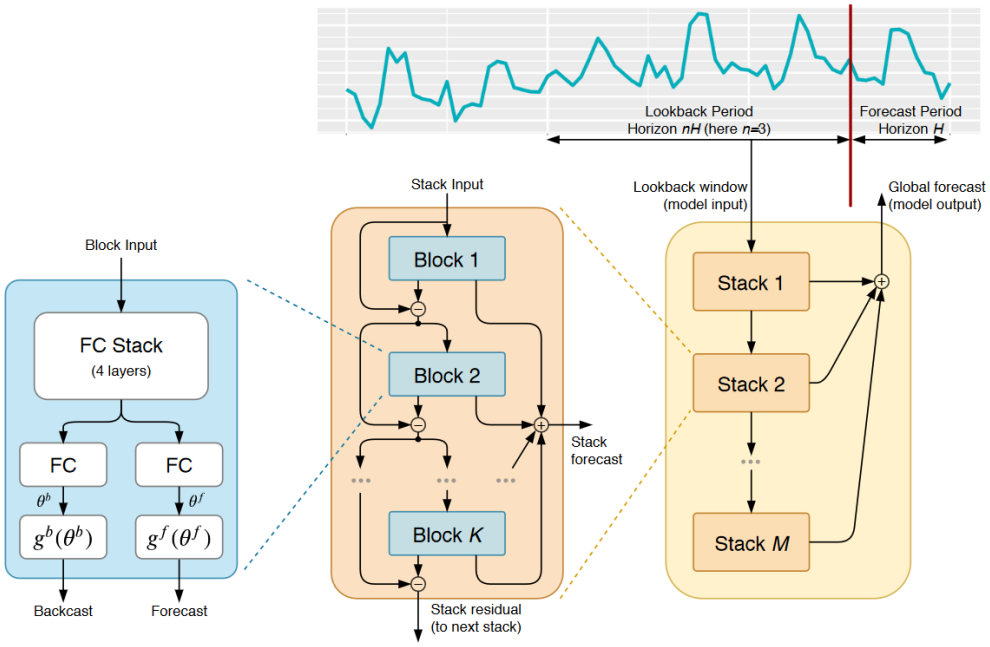


Figure 2.12: N - Beats Architecture [53]

Chapter 3

Design of Proposed Scheme and Approach

3.1 Overview

In decentralized networks, all nodes in the network are not aware of each other's transmissions. Therefore, no node in the network can keep records of all the transmissions going through the network. Thus, a network monitoring tool is appropriate to record all the transmission outcomes of all the nodes in a network. Even though there are some powerful network monitoring tools like Solarwinds network performance tool and Datadog network performance tool, significant changes in the nodes are required to utilize them more efficiently. Concurrently, a minimal modification is desired in the network nodes and insignificant or least possible burden on nodes is preferred. All the criteria are pointing towards employing a cloud to monitor the network. To meet the criteria of the least possible burden on the network nodes, acknowledgement packets are identified as the best option to use as a means of communication between nodes and the cloud. This scheme that employs the cloud as a monitoring tool and acknowledgement packets as a means for communication is called Cloud ACKnowledgement Scheme (CACKS).

As a solution to the challenge mentioned in the problem statement, CACKS is designed to monitor and provide a particular type of cloud service as a backup to the nodes in a network. CACKS uses specially formed ACK and NAK packets as means to monitor the

data communication activity in a network. CACKS utilizes the location of the data packet and the status of the transmission for monitoring nodes in that particular network.

In this research, we focus on providing backup services to nodes in a situation in which a node has to reforward the data packet. This approach also aims to identify the decent nodes in a network by gauging the performance of the nodes and avoid the data communication through the nodes with the least performance. The nodes with the least performance are more likely to be selfish nodes and malicious nodes [37]. The performance of a network is greatly dependent on the individual performance of the nodes. All the nodes in the network must work with each other to deliver the best functioning network. So it is essential to identify the selfish nodes in the network to improve the overall performance of the network [63]. A node is declared as a selfish node if it aims to benefit itself. It practices selective communication or no communication of data packets to conserve energy. These selfish nodes make a network sluggish and slow. Similarly, malicious nodes are described as the nodes with the following characters

1. Packet Drop
2. Delay
3. Link Break
4. Message Tampering
5. Denying from Communication
6. Fake Routing
7. Node not Available
8. Stealing Information
9. Session Capturing

3.2 Assumptions and Notations

In this section, we describe all assumptions that were put forward to develop our proposed scheme as well as our proposed scheduler. Moreover, we list all notations used with their corresponding definitions in Table 3.1.

3.2.1 Assumptions

- The Cloud is within the range of the network.
- The cloud is neither malicious nor faulty.
- The receiving and the transmission range are the same for all Virtual Machines (VMs) in the cloud. Also, all VMs are working correctly.
- Each VM can accept and process only one task at a time.
- All VMs are identical to each other.
- All tasks are considered as Non-preemptive, where the task is given resources until it gets terminated or it reaches a waiting state.
- Subtasks are not allowed to split to compute in other VMs.
- All the tasks are computationally intensive and mutually independent, while the task subsets are collectively dependent.

3.2.2 Notations

Table 3.1: Description of Notations Used in Proposed Scheme

Symbol	Description
VM	Virtual Machine
N_{tasks}	No of tasks waiting in the pool
T_x	Task ID
V_j	Virtual Machine ID of j^{th} VM

Continuation of Table 3.1

Symbol	Description
a_i	Acknowledgement ID of i^{th} packet
N_{ack}	Number of Acknowledgements in a task T_x
ST_x	Size of the task T_x
Spd_j	Speed of VM of V_j
E_j	Energy consumption per unit time of V_j
TCT_j	Time to compute a task T_x
ECa_i	Expected Communication time of an acknowledgement ai
EPa_1	Expected time to process an acknowledgement ai
h	Hop count of a task T_x
f_j	Frequency of a VM V_j
V_{max}	Maximum number of VMs allowed
DT_x	Deadline for a task T_x
B_m	A Batch of tasks

3.3 CACKS Design

Among the different types of data packets, two types of packets are altered and implemented in the CACKS approach for data collection. They are Acknowledgement packets (ACK) and Negative Acknowledgement packets (NAK). The flags field indicates the nature of the transmission. To distinguish each packet type, Table 3.2 indicates the packet types and its flags or options.

Table 3.2: Data Packet Types and Data Packet

Packet Type	*General*	*Acknowledgement*	*NegativeAck*
Packet Flag or Option	*PSH*	*ACK*	*NAK*

3.3.1 CACKS Modified Acknowledgement Packets

A TCP ACK packet holds the information about the transmission of a particular data packet in a series of data packets [57]. The source *Node A*ddress, destination *Node A*ddress, and sequence number included in the packet are of special importance. ACK packets do not hold any payload which makes the packet light to transmit. An ACK number is also included along with flags, window size, and checksum to represent some of the fields required in a TCP/IP transmission.

However, just the above-mentioned data about transmission is insufficient to capture the big picture of the transmissions. Therefore, along with the standard fields, a modified ACK packet incorporates two more address fields as shown in the Figure 3.1. The new fields are current *Node A*ddress, which contains the address of the node holding the data packet, and forwarded *Node A*ddress, which contains the address of the neighboring node to which the data packet will be forwarded. These extra fields allow the cloud to monitor the status of the transmissions using CACKS. The modified ACK packet provides the whereabouts of a particular data packet in transmission.

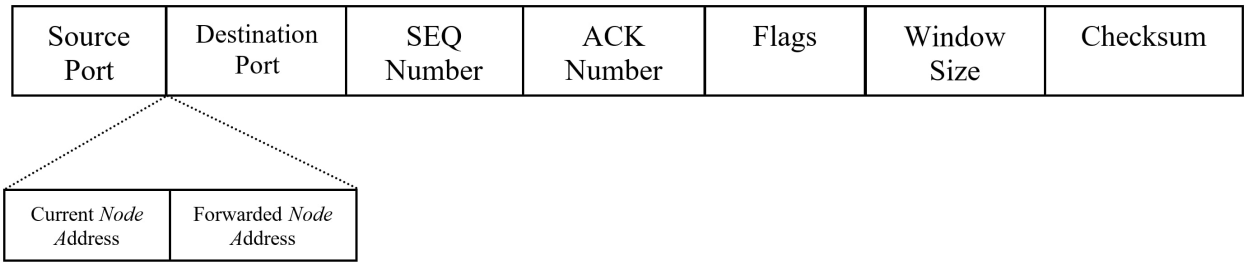


Figure 3.1: Modified ACK Packet

Whenever the destination node receives a corrupted packet, it responds with a NAK indicating that the packet was received in a corrupt state. Simultaneously, the NAK packet also represents the packet droppings when received by the sender. This signal prompts the sender to retransmit the packet. Another situation is when the sender node misses an ACK packet, it indicates packet loss and the destination node did not receive a data packet at all. The sender node will retransmit lost data packets after a certain time. Therefore, including an additional address field that carries attempted forwarding *Node A*ddress gives away the intermediate location of the data packet as shown in the Figure 3.2. This feature can help

to identify the node responsible for packet loss. This altered NAK packet holds information about a node that does not respond to a connection that has already been established [45].

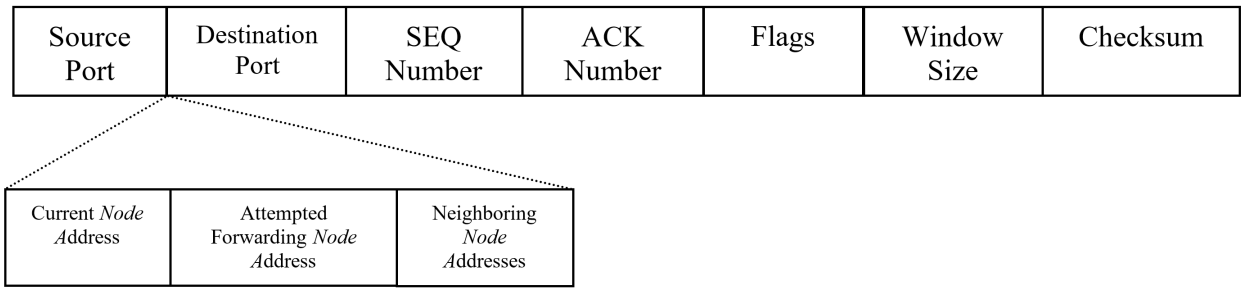


Figure 3.2: Modified NACK

Along with the modified packets, $P_{initial}$,$P_{request}$ and P_{INFO} are three compact packets that are very significant in this approach. Firstly, $P_{initial}$ is used by the sender node to begin a transmission. Secondly, $P_{request}$ is used by the nodes to request data from the cloud. Finally, P_{INFO} holds the response to the information requested by the node.

3.3.2 CACKS Architecture

To monitor and collect data about each node's transmission status in a network, this scheme deploys a cloud. Unlike regular TCP/IP control flow as shown in Figure 3.3, where a sender node initiates a data packet P_{data}'s transmission and it takes place through a route with few intermediate nodes *Node A*, *Node B*, *Node C* and *Node X* to reach a destination node. The response acknowledgement packet P_{ack} is generated by the destination node only. This acknowledgement packet takes the same route as the data packet to reach the sender node. Through this approach, it is not possible to monitor the complete network.

In the furtherance of the cloud to monitor the network, the regular network control must be adjusted. Figure 3.4 demonstrates the approach of the cloud to monitor the network. Assuming that a packet route is established between Sender and Destination with *Node A*, *Node B*, *Node C*, *Node X* as intermediate nodes, the sender desires to initiate the transmission of data packets. To commence the transmission, senders must secure a connection with the cloud so that cloud is aware of the transmission. So as to connect to the cloud, Sender sends a $P_{ack(initial)}$ to the cloud. This $P_{ack(initial)}$ is formed with the primary information like sender

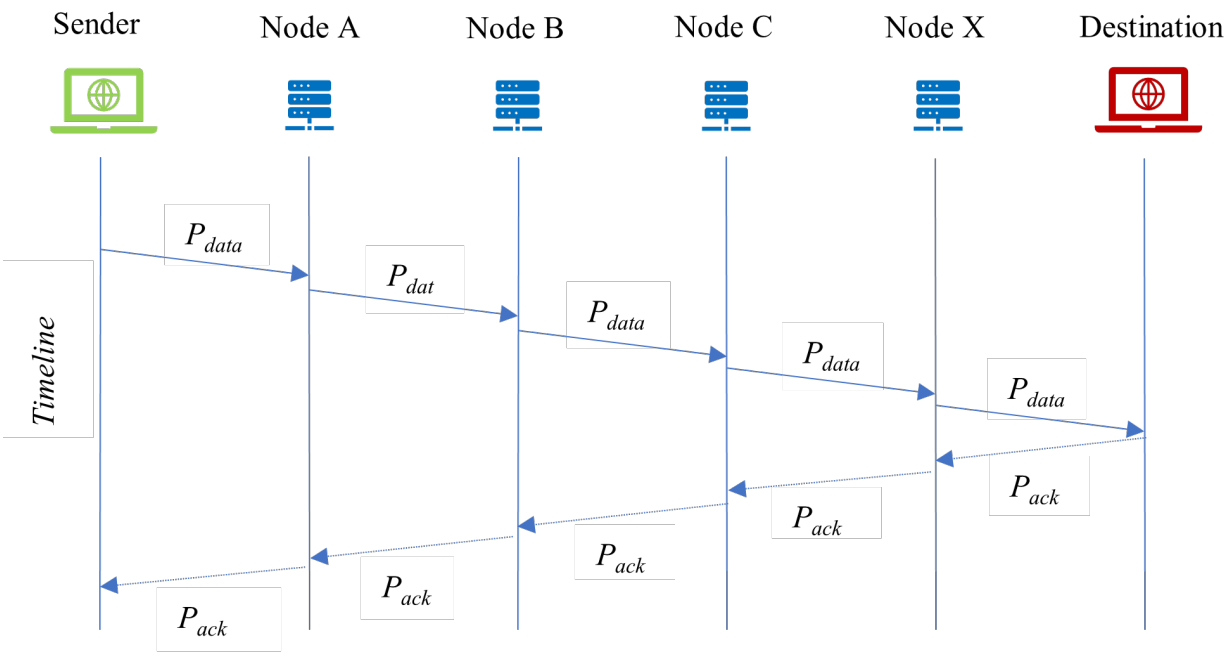


Figure 3.3: Normal Network Control Flow

*Node A*ddress and destination *Node A*ddress. This packet provides necessary information for the cloud to set the monitoring procedure in motion. Once the setup for monitoring procedure is complete, Cloud replies to the sender node with a *Ping()*. This action signals the sender to proceed with the transmission of the data packet P_{data}. So the Sender node starts pushing data packets to the neighbour *Node A* and transmits an acknowledgement packet $P_{ack(A)}$ to the cloud. Once *Node A* completes its initial formalities, it formulates a $P_{ack(A)}$ that consists of current *Node A*ddress that is *Node A*'s address and forwarded node's address which is *Node B* in this scenario. Acknowledgement packet $P_{ack(A)}$ provides the information of the location of the data packet to the cloud, which is *Node B*. Similarly, after P_{data} reaches *Node B*, an acknowledgement packet $P_{ack(B)}$ is formulated and transmitted to cloud. This process repeats until the data packet P_{data} reaches its destination node. The formulated $P_{ack(D)}$ is forwarded to Cloud. When the cloud receives the final acknowledgement packet, the cloud assumes that the P_{data} reached its destination. Simultaneously the sender needs to know the status of a data packet. For this reason, cloud forwards the acknowledgement packet $P_{ack(D)}$ to the sender node. This process after receiving the *Ping()* from cloud repeats

at a larger scale for every data packet in the sequence. This continuous acknowledgement packet transmission to the cloud keeps it in the loop of the transmissions across the network and monitors the status of the nodes in the network.

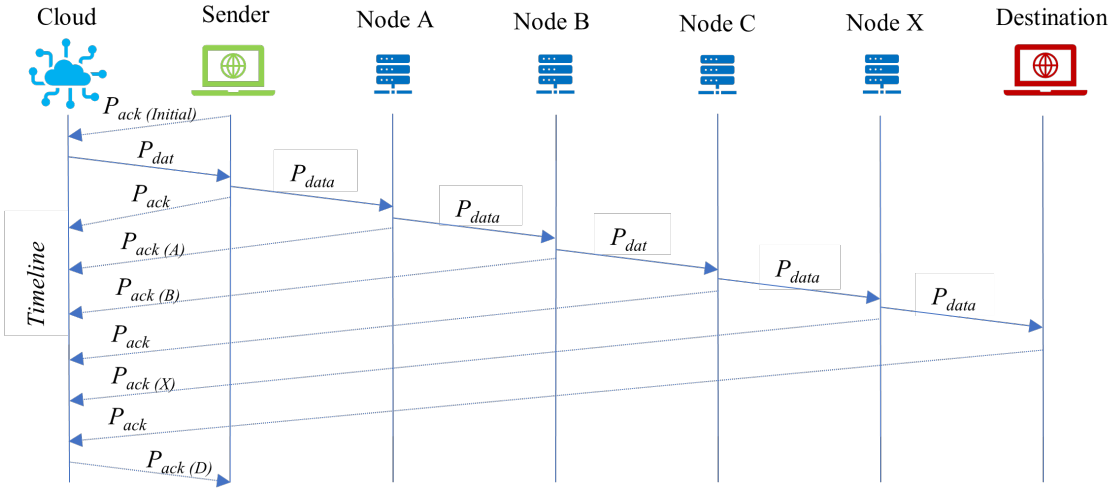


Figure 3.4: Cloud Acknowledgement Scheme Control Flow

In the scenario where there is no packet loss, the cloud acts as a simple monitoring tool. Cloud and each of the node's roles are described in the developed algorithms. There are three algorithms named Cloud-site Algorithm, Sender-site Algorithm, and Intermediate Node-site Algorithm. The sender-site Algorithm explains the procedure followed by the sender node to initiate the packet transmission process. The Cloud-site Algorithm provides the detailed description of the steps taken by the cloud after receiving $P_{ack(initial)}$ from the sender node. The Intermediate Node-site Algorithm explains the protocol that an intermediate node follows whenever a packet transmission occurs through it.

The following algorithm is the Sender-site algorithm. This algorithm begins when the sender node wishes to communicate with the destination node. Sender node creates a $P_{ack(initial)}$ that contains the addresses of itself and destination node. A timer starts and expects the cloud's response within the predefined time PT. If the cloud does not respond, $P_{ack(initial)}$ is retransmitted. If the sender receives the response from the cloud in the form of $Ping()$, it starts creating the data packet P_{data}. This holds the data along with the header information like sender address, destination node address, and forwarding node address which

is *Node A*. Once the data packet is transmitted to *Node A*, the sender formulates the acknowledgement packet with *Sender Node* address and *Destination Node* address. If the sender node receives any negative acknowledgement packet, it retransmits the data packet with that particular sequence number.

Algorithm 1: Sender-site Algorithm

1. *Sender Node* creates $P_{ack(initial)}$ // *Formulate the initial packet*
   (a) Include *Sender Node* address to $P_{ack(initial)}$
   (b) Include *Destination Node* address to $P_{ack(initial)}$
2. *Sender Node* sends $P_{ack(initial)}$ to the Cloud
3. Count = 0, MAX = Set to Predefined Time (PT)
4. Increment Count by 1
   (a) If (Count < PT & *Node A* received *Ping* from Cloud) Then
   (b) Proceed
   (c) Else-If (Count >= PT & *Node A* did not receive *Ping* () from Cloud) Then
   (d) Resend $P_{ack(initial)}$ and go to Step 3
5. *Sender Node* creates P_{data} // *Create a regular data packet along with the following additional data*
   (a) Include *Sender Node* address to P_{data}
   (b) Include *Destination Node* address to P_{data}
   (c) Include *Node A* address to P_{data}
6. *Sender Node* forwards P_{data} to *Node A*
7. *Sender Node* creates $P_{ack(S)}$
   (a) Include *Sender Node* address to $P_{ack(S)}$ //source address

(b) Include *Destination Node* address to $P_{ack(S)}$ //destination node address

(c) Include *Node A* address to $P_{ack(S)}$ //forwarding node address

(d) Include *Sender Node* address to $P_{ack(S)}$ //current node address

8. *Sender Node* forwards $P_{ack(S)}$ to Cloud

9. Proceed with next data packet

10. If (*Sender Node* receives $P_{ack(D)}$) from Cloud

    (a) Proceed

11. Else if *Sender Node* receives *NAK* from cloud

    (a) Repeat from step 5

The cloud-site algorithm describes the steps the cloud must follow for monitoring and collecting data about the node's transmission statuses. This process begins when cloud receives $P_{ack(initial)}$ from *Sender Node*. After reading the header, it initiates a memory buffer after validating the data packet and stores the acknowledgement packets coming from *Sender Node* and intermediate nodes. If the $P_{ack(initial)}$ is corrupted, cloud sends a *NAK* to the *Sender Node*. Cloud sends a *Ping()* to signal *Sender Node* to proceed with transmitting data packets. Cloud starts a timer and waits for the acknowledgement packet from *Sender Node*. If cloud do not receive $P_{ack(S)}$, a *NAK* is sent to *Sender Node*. Max count for each acknowledgement packet from each of the intermediate nodes is set to Predefined Time *PT*. For successful transmission from each of the intermediate nodes, the cloud rewards the node by adding 1 to its node value. If the cloud does not receive an acknowledgement packet from the forwarded node, the cloud subtracts 1 from its node value. The same step applies if the cloud receives an invalid or corrupt packet. If the cloud receives a *NAK* from any intermediate node, the cloud figures out that the forwarding node is not responding and the unresponsive node loses 1 from its node value.

Cloud forecasts the likelihood of the next node's transmission is a success or failure using machine learning algorithms and the values are shared with the intermediate node. Forecast values are shared with the intermediate node that is affected by the forwarding node's packet

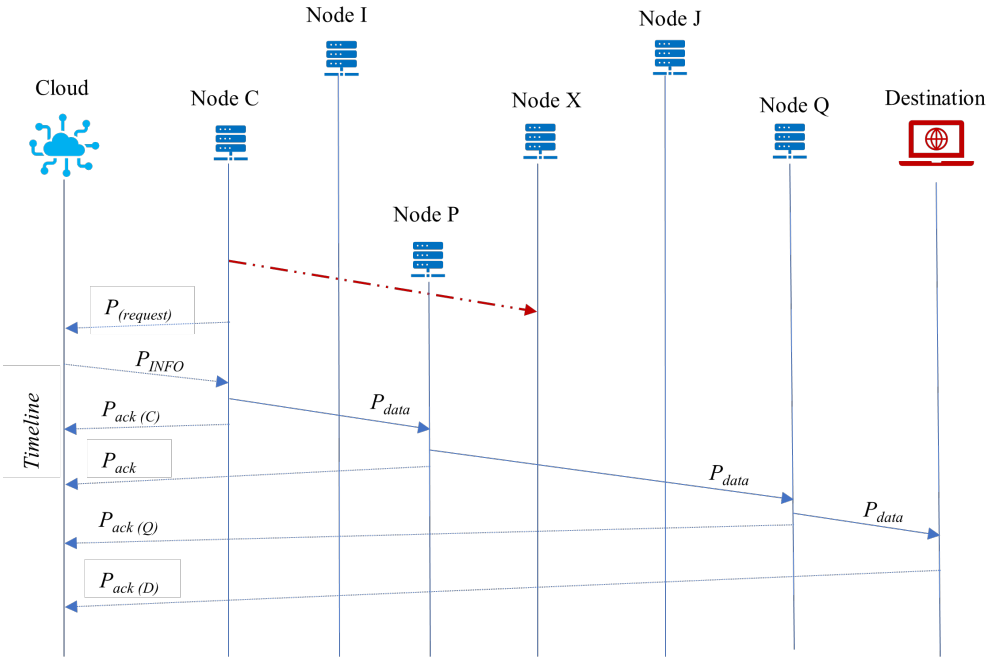


Figure 3.5: Failed Intermediate Node Transmission Scenario

drop issue. Another condition for sharing the forecast values is that affected intermediate node must request a cloud with a $P_{request}$ as shown in the Figure 3.5. This special $P_{request}$ holds the intermediate node's neighbor nodes addresses. Packet P_{INFO} is created with the forecast values and sent to the affected intermediate node. The *NAK* transmitted by the affected intermediate node is forwarded to the *Sender Node*. Finally, if the cloud receives $P_{ack(D)}$ from *Destination Node*, it is forwarded to the *Sender Node* once it is validated.

For example, as shown in the Figure 3.5, *Node X* is unresponsive and disconnected from the packet route and represented by a red arrow. Therefore, *Node C* sends a $P_{request}$ to the cloud. After cloud forecasts the node values of its neighbors *Node P* and *Node I*, a P_{INFO} is created and transmitted to the affected *Node C*. Later, *Node C* decides to continue data packet transmissions through *Node P* and then through *Node Q*. Once the acknowledgement packet $P_{ack(D)}$ reaches cloud from the destination node, the process concludes.

Algorithm 2: Cloud-site Algorithm

1. Cloud receives $P_{ack(initial)}$ from *Sender Node // Start of processing sender node*

   (a) Parse $P_{ack(initial)}$

(b) Initialize a buffer memory slot

(c) Validate the $P_{ack(initial)}$

i. IF ($P_{ack(initial)}$ is valid) Then

ii. Store $P_{ack(initial)}$ in the buffer

iii. Else

iv. Send *NACK* to *Sender Node*

(d) Process the contents of $P_{ack(initial)}$ // *End of processing sender node*

2. Ping *Sender Node* by sending a *Ping()*

3. Count = 0, MAX = Set to Predefined Time (PT)

4. Increment Count by 1 // *Start of processing one intermediate node*

(a) IF (Count $\leq PT$) Then

i. Wait until PT expires

ii. IF $P_{ack(A)}$ Received from *Node A*

A. Parse $P_{ack(A)}$

B. Validate $P_{ack(A)}$

C. IF ($P_{ack(A)}$ is valid) Then

D. Store in the buffer

E. Add 1 and then update the value of *Node A* in the Index

F. Else

G. Send *NACK* to *Sender Node*

H. Subtract 1 and the update the value of *Node A* in the Index

I. Process $P_{ack(A)}$

J. Store in the buffer

iii. Else if *NACK* Received from *Node A*

A. Process *NACK*

B. Subtract 1 and then update the value of *Node B* in the Index

C. IF $P_{(request)}$ received from *Node A*

D. Forecast the value of neighbouring nodes *I, J, K, L* //As shown in Figure 3.5

E. Create P_{INFO} // contains forecasts of *Node A*'s neighbouring nodes *I, J, K, L*

F. Send P_{INFO} to *Node A*

iv. Else

A. Wait

(b) Else

i. Send *NACK* to *Sender Node // End of processing an intermediate node*

5. $P_{ack(D)}$ received from *Destination Node // Start of processing destination node*

6. Parse $P_{ack(D)}$

7. Validate $P_{ack(D)}$

(a) IF ($P_{ack(D)}$ is valid) Then

(b) Store in the buffer

(c) Else

(d) Send *NACK* to *Sender Node* and *Destination Node*

(e) Subtract 1 and the update the value of previous intermediate node in the Index

8. Process $P_{ack(D)}$ *// End of processing destination node*

9. Forward $P_{ack(D)}$ to *Sender Node*

The intermediate nodes require to follow some regulations as formulated in the following Intermediate Node-site algorithm. Once the route is established among the nodes, intermediate nodes expect the data packets. Once the intermediate node receives the data packet

P_{data} it reads the destination address in the packet header. It simply forwards the data packet to the next neighbor in the routing table. After transmission of the data packet, the neighboring node must create the acknowledgement packet with *Sender Node* address, *Destination Node* address, and its own address. This acknowledgement packet is then transmitted to the cloud. Whenever the intermediate node faces a challenge with an unreachable or unresponsive issue with the next neighbor in the routing table, it can send a $P_{request}$ with the address of all the potential neighbors. Simultaneously, it needs to send a *NAK* packet after dropping the data packet P_{data}. Afterward, when the cloud sends the forecast node values of each neighbor, it can select one among them to continue the data packet transmission

Algorithm 3: Intermediate Node-site Algorithm

1. *Node A* Received P_{data} from *Sender Node*
   (a) Parse P_{data} *to check for Node B address*
   (b) Read destination address
2. If (*Node B* is reachable) Then
3. Forward P_{data} to *Node B*
4. Create $P_{ack(A)}$
   (a) Include *Sender Node* address to $P_{ack(A)}$ //source node address
   (b) Include *Destination Node* address to $P_{ack(A)}$ //destination node address
   (c) Include *Node A* address to $P_{ack(A)}$ //currrent node address
   (d) Include *Node B* address to $P_{ack(A)}$ //forwarded node address
5. *Node A* sends $P_{ack(A)}$ to Cloud
6. Else If (*Node B* is not reachable) Then
7. Drop P_{data}
8. Create *NAK*

(a) Include *Sender Node* address to *NAK* //source node address

(b) Include *Destination Node* address to *NAK* //destination node address

(c) Include *Node A* address to *NAK* //current node address

(d) Include *Node B* address to *NAK* //attempted forwarding node address

9. Send *NAK* to the Cloud

(a) Discover neighboring nodes // *Nodes I, P* are neighbors (as shown in Figure 3.5)

(b) Create $P_{request}$

i. Include *Node A* address to $P_{request}$ //requesting node address

ii. Include *Node P* address, *Node Q* address to $P_{request}$ //neighboring node addresses

(c) Count = 0, MAX = Predefined time PT

(d) *Node A* sends $P_{request}$ to the Cloud

(e) Increment Count by 1

i. IF (Count $\leq PT$) Then

A. Wait until PT expires

B. IF P_{INFO} Received from Cloud

C. Parse P_{INFO}

D. Validate P_{INFO}

E. IF (P_{INFO} is valid) Then

F. Read P_{INFO}

G. Select node with the best forecast value

H. Forward P_{data} to *Node K*

ii. Else

A. Send *NACK* to Cloud

10. Else

(a) Forward P_{data} to *Node B*

3.4 Energy Aware Hybrid Scheduling Strategy

3.4.1 Overview

In this section, the reseacher proposes a new scheduling algorithm for the cloud. The main aim of this scheduling approach is to sustain the ephemeral nature of the CACKS in a node network. Generally, scheduling algorithms are classified into two types. The first type is referred as an independent scheduling algorithm. The second type is referred as a workflow scheduling algorithm. Our proposed scheduling algorithm falls within or is classified as an independent scheduling algorithm. This is because each acknowledgement task is autonomous. Due to the comprehensive nature of the CACKS scheme, the scheduler consists of two working algorithms, namely: variable time algorithm and variable frequency algorithm, as shown in Figure 3.6. The scheduler toggles between these two algorithms based on certain circumstances, which we explain next. There are also other components in the scheduler that help in supporting the proposed schedule algorithm. This includes a *VM* model that is designed for virtual machines to optimize the energy consumption and performance, and a meta-scheduler to organize the tasks as they arrive at the task-pool [61]. Each component in this scheduler is explained in the following sections.

3.4.2 Meta-Scheduler

The meta-scheduler illustrated in Figure 3.6 is the virtual scheduling assistant to help the scheduler minimize the time by preprocessing tasks to reduce the processing time required by the scheduler. The main idea of having a meta-scheduler is to have an efficient scheduling approach, especially when the lined tasks are picked from a broader pool. The meta-schedule is a combination of two processes. The first process is to create a meta-data and tag the data to the task. The second process is a continuous cycle that runs forever. During the second process, the meta-scheduler refreshes the task pool by updating the number of tasks available in the task pool and ready for execution. This process helps the meta-scheduler to switch between the two developed algorithms when needed. The processes and control required to handle tasks using the proposed meta-scheduler are illustrated in Figure 3.7.

When an initial acknowledgement, a_0, arrives at the taskpool, a new task T_x is created and

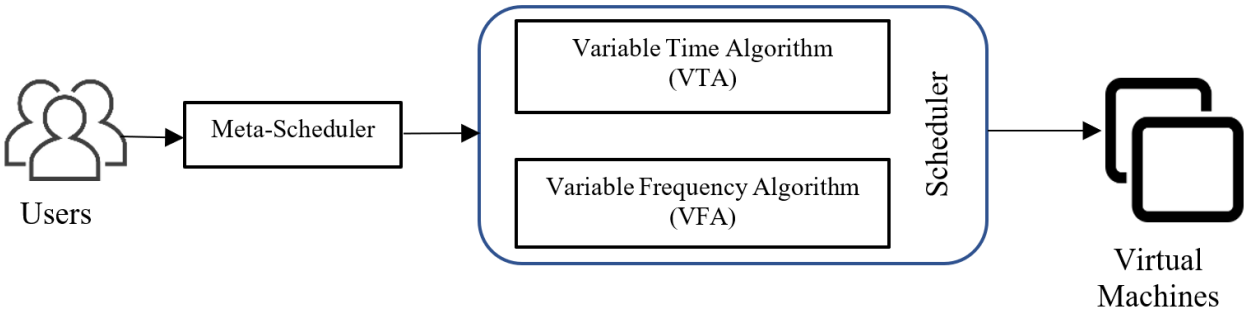


Figure 3.6: Scheduling Environment for the Cloud

remains in the task-pool until the task is assigned to a virtual machine. All acknowledgement packets *(a0, a1, a2,. . . , ai)* of a particular task Tx are collected in the virtual machine and processed. While meta-scheduler chooses to operate in a variable time algorithm, it maps the tasks T_x ($x = 1, 2, 3, \ldots, n$) to a specific VM V_j ($j = 1, 2, 3, \ldots, m$). The mapping is done using a variable time algorithm to conserve energy as much as possible by mapping the massive tasks to a *VM* with an optimum CPU frequency $f_j^{opt}(f_j^{min} \leq f_j^{opt} \geq f_j^{max})$. The size of task T_x is ST_x, which is directly proportional to the number of hops h. Since it is possible to retrieve h from Forward Information Base (FIB) with the total number of acknowledged packets Nack [25], task T_x is always ahead by the value of 2 with respect to h. Categorically, a task consists of multiple acknowledgement packets a_i ($i = 0, 1, 2, 3, \ldots, n$) that cloud collects after a sender initiates packet transmission. Therefore, a sender S always sends two acknowledgement packets (a_0, a_1) in the CACKS, where the size of these packets can be computed using Equations 3.1 and 3.2.

$$N_{ack} = h + 2 \tag{3.1}$$

$$ST_x = N_{ack} \tag{3.2}$$

The size of the task ST_x is required to determine the expected frequency of task T_x. The TFT_x of a task can be calculated using Equation 3.3. This information is required to feed the current working algorithm in the cloud.

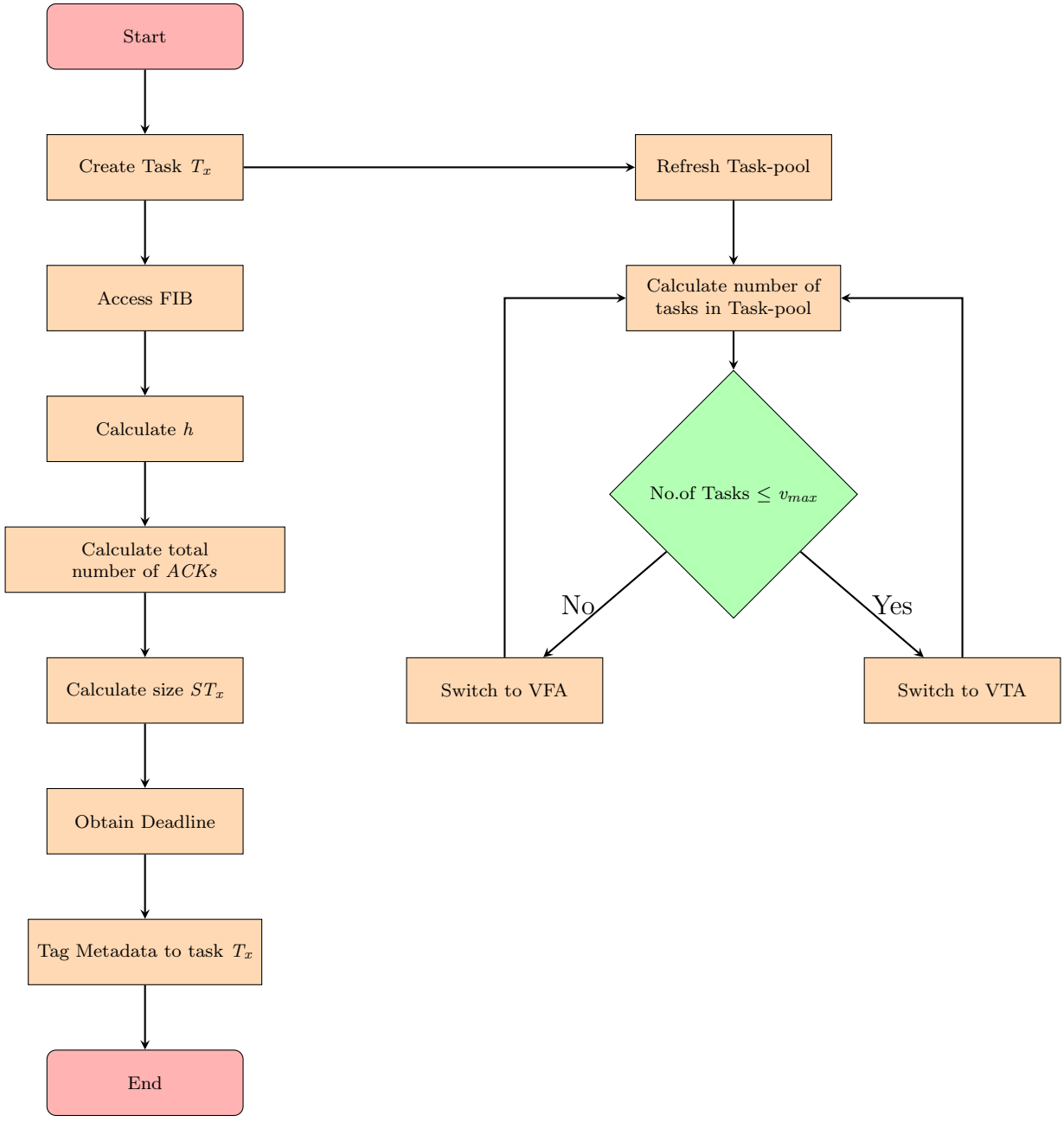


Figure 3.7: Data and Control Flow of Meta-Scheduler

$$TFT_x = \sum_{i=0}^{n}(ECa_i + EPa_i) + \text{VM } \textit{Time required to respond with the outcome to sender S} \tag{3.3}$$

If we consider the situation when ($N_{tasks} \leq V_{max}$) at the initial stages, then the active algorithm is Variable Time Algorithm. Once the number of tasks N transcends the available virtual machines V_{max}, the meta-scheduler switches to the variable frequency algorithm. After the sender initiates packet transmission, the cloud collects multiple acknowledgement packets a_i (i = 0, 1, 2, 3,. . . , n), at the same time there are processes running in parallel within the meta-scheduler.

3.4.3 Scheduling Model

Variable Time Algorithm (VTA)

The variable time algorithm (VTA) algorithm is intended to emphasize energy conservation. Variable time algorithm is a segment of our proposed work, providing an energy-efficient method. This algorithm operates when the total number of tasks N_{num} is falling short of the total number of available virtual machines, V_{max}. In this case, as soon as a task arrives in the task-pool, it will be assigned to a running virtual machine. So, the total number of tasks in the task-pool waiting to be executed should be empty. VTA also suspends the idle *VM*s whenever there is no activity in a particular *VM* (i.e., $N_{num} < V_{max}$). It maintains the suspended state as long as there is no new task waiting to be executed in the task-pool. At the same time, *VM* maintains the optimum CPU frequency at minimum (i.e., $f_j^{min} = f_j^{opt}$) considering f_j^{min} = 38 percent of f_j^{max} [29].

Whenever a new task arrives in the task-pool, it is assigned to an idle *VM* by the VTA algorithm. The *VM* changes its state from idle to active as soon as the task is assigned. If there is an active *VM* that just accomplished its task, the new task is assigned to that active *VM*. Equation 3.4 is used to ensure that there is a fixed amount of time (time in the active-idle state), in which the *VM* will be active but idle. During this time the *VM* is ready to deploy if a new task swoops in the task pool; beyond this time *VM* suspends its activity and shuts down.

$$Time\ in\ Active\ -\ Idle\ state = \frac{EnergyRequiredto(Turnoff + Turnon)anidleVM}{EnergySpentonanIdleVMperunittime} \quad (3.4)$$

The processes and the functions that are performed inside a single $VM\ \ V_j$ are described in VTA Algorithm. The optimum frequency f^{opt} is set to a minimum frequency f^{min} to conserve energy for any taken task in the virtual machine V_j. The TFT_x is calculated by meta-scheduler and tagged to the task and the resulting value is used as the deadline for a particular task T_x. Time to compute TCT_x starts and increases at a steady rate, and it keeps on increasing until it reaches TFT_x. A task is given the required resources to compute until TCT_x reaches TFT_x. When TCT_x exceeds TFT_x, the task time out is sent to the sender, a report is archived, and the task is killed. This information is related to this sub-task and a report is created with this information.

Algorithm 4: Variable Time Algorithm (VTA) in a $VM\ \ V_j$

1. Set frequency $f_j = f_{min}$
2. Foreach $Vj = (V1,....,Vn)$ do
   (a) Fetch task T_x
   (b) Retrieve meta-data
   (c) Process meta-data
   (d) Count TCT_x begins; TCT_x ++
   (e) If ($TCT_x <= TFT_x$)
      i. Foreach Acknowledgement packet a_i (a_1,....,a_n) in T_x do
         A. Receive a_i
         B. Process a_i
         C. Finish task T_x
      ii. End If(NACK received)
         A. Send NACK to sender node S

3. Until If (TCT_x >TFT_x)

   (a) Send Time_Out;

   (b) Send Report

   (c) Archive Report

   (d) Kill the task

Variable Frequency Algorithm (VFA)

With our developed variable frequency algorithm (VFA), we focused more on the performance by adjusting energy consumption. This algorithm elevated the performance of the cloud processor by utilizing the batch scheduling concept [56]. This concept is reformed to match the requirements of this scheme. The acknowledgement scheme requires a considerable amount of waiting time in which the cloud receives the acknowledgement packets from nodes in the network. So, it is pointless to allocate a large number of resources to all the tasks. Batch scheduling is the optimum strategy to conserve energy while providing great performance. The VFA algorithm is triggered when the number of tasks in the task-pool is waiting to be executed, N_{tasks}, surpasses the total number of VMs, i.e. V_{max}. Algorithm 5 explains how VFA is processed. It is uncomplicated to create a batch B_s, because the number of tasks is more than the maximum number of VMs. Thus, the batch size is equal to the maximum number of VMs ($B_s = V_{max}$). VFA is constrained with time deadline for task T_x; thus, it is a significant factor in deciding the optimum frequency f^{opt} of a certain virtual machine. Equation 3.5 is used to calculate the f^{opt} of a single task.

$$f^{opt} = \frac{ST_x}{DT_x - Start\ time\ of\ Task_x} \tag{3.5}$$

The tasks which arrive in the task pool are taken into batches as soon as they arrive, so the first come first serve (FCFS) approach is used to serve the upcoming batches. Once a batch is filled up, the next batch is created. Each batch has a fixed amount of time in which it should complete. Table 3.3 represents the VFA schedules tasks, using the BatchTask-VM (BT-V) matrix with m batches, n tasks, and n *VMs*.

Table 3.3: BT-V Matrix

BT-V	V_1	V_2	...	V_n
B_1T_n	$B_1T_1V_1$	$B_1T_2V_2$	...	$B_1T_nV_n$
B_2T_n	$B_2T_1V_1$	$B_2T_2V_2$	...	$B_2T_nV_n$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
B_mT_n	$B_mT_1V_1$	$B_mT_2V_2$	...	$B_mT_nV_n$

The function of VFA is represented in Algorithm 5. All the tasks are waiting to be computed as batches. A batch contains v_{max} the number of tasks because all the tasks in a batch are computed at once. Therefore, the maximum number of tasks that can be executed at a time is equal to the maximum number of virtual machines. An empty path list is created to map the tasks to *VMs* available. The time for each batch varies according to the task with the nearest deadline, T_{max}, which is assigned to the task with the shortest deadline in the batch. All the tasks in a particular batch should be completed at the same time, that is T_{max}. The optimum frequency is calculated using Equation 3.5. The time to compute starts and increases steadily at a uniform rate. The task is assigned resources until TCT_x reaches T_{max}. After this case, a time-out message is sent to the sender. Then, a report with all the events during communication is created and sent to the sender, and the task is killed.

Algorithm 5: Variable Frequency Algorithm (VFA)

1. Input: A FCFS task list in Task-pool = $[T_1,\ldots,T_n]$
2. output: A Path List PL
3. $PL = \phi$; //Empty path list
4. Set MAX = V_{max}
5. Repeat for each T_x in Task-pool
   (a) For each B_m in $[B_1,B_2,\ldots,B_n]$
      i. Add T_x to batch B_m //Task is included in the batch
      ii. Remove T_x from Task-pool

(b) End for

6. Until B_m = MAX
7. Add B_m to PL // Batch is mapped with path list
8. Find least TFT_x in B_m //Find task with least TFT_x in the Batch
9. Read A path list PL
10. Repeat for each $PL = [P_1,\ldots,P_n]$
    (a) For each $V_j = [V_1,\ldots,V_n]$
        i. Set $T_{max} = T_x$
        ii. Inset $PL \mapsto V_j$
        iii. Calculate f^{opt}
    (b) End for
    (c) Initialize TCT_x to 0
    (d) While ($TCT_x \leq TFT_x$) Do
        i. For each Acknowledgement packet $a_i(a_1,a_2,\ldots,a_n)$ in T_x do
            A. Receive a_i
            B. Process a_i
            C. Finish task T_x
        ii. End for
        iii. Initialize TCT_x to 0
        iv. While ($TCT_x \leq TFT_x$) Do
        v. For each Acknowledgement packet $a_i(a_1,a_2,\ldots,a_n)$ in T_x do
            A. Receive a_i
            B. Process a_i
            C. Finish task T_x
        vi. End for

vii. if ($NACK$ received)

A. Terminate

B. Send $NACK$ to sender node S

(e) Read TCT_x

(f) End While

11. Until ($TCT_x > TFT_x$)

12. Send Time out

13. Create a report

14. Send the report

15. Archive report

16. Kill the task

3.4.4 Transition Time

Since our approach provides a hybrid scheduling algorithm, this section explains the steps involved in the schedule to switch from VTA to VFA and vice versa. In the former, the meta scheduler allows the task pool to expand with the arrival of new tasks. During this process, the meta scheduler notifies VTA to switch to VFA. The VTA boosts its f^{opt} to f_{max} to finish all the remaining tasks that are running in the cloud at the switching moment. In the latter, when the scheduler does not have enough tasks for creating a batch, the meta scheduler signals VFA to switch and let VFA finish its current tasks.

3.4.5 Energy Model

This section explains the energy consumption of the VM for one task in CACKS for a node network. Currently, data centers consume around 2 percent of the total power consumption around the world, and consumption is expected to reach 8 percent in the coming years [26]. Within the data center, feeding power to various servers required 40 percent of the total

energy consumed by the data center. At the same time, around 38 percent of total power is invested in maintaining the cloud cooling system [44]. Figure 3.8 is the global trend of energy consumption by the IT industry [36].

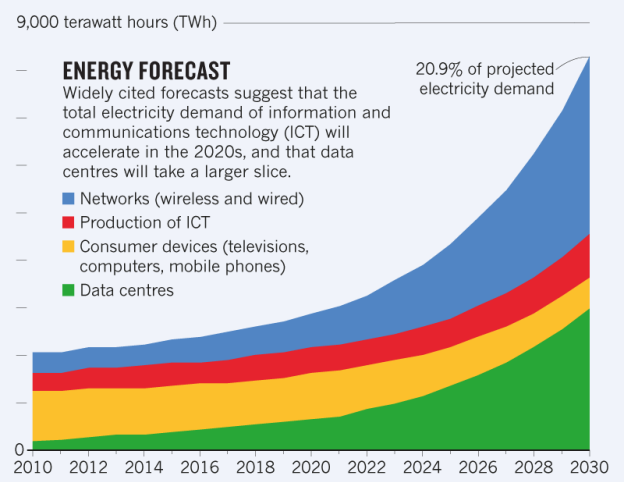


Figure 3.8: Energy Utilization Forecast [36]

Conserving energy makes a significant difference in both the environment and on the cloud service provider. The aggregate energy exhausted by all tasks is huge. This expenditure does not include energy spent while the *VM* is idle or while switching between VTA and VFA. The total energy is calculated using Equation 3.6.

$$TotalEnergy = Computationenergy + Communicationenergy + Energytocool \tag{3.6}$$

The cooling energy is the consumed energy spent on cooling the system during the processing time of task T_x. The power of a computing machine is measured in terms of energy per instruction *(EPI)*. It is the average of the total amount of energy spent on task x and calculated using Equation 3.7[31].

$$EPI = \frac{Joules(J)}{Instruction(I)} W/IPS \tag{3.7}$$

The computation energy is the energy drained by the *VM* on task T_x. Equation 3.8 is used to compute the energy spent by a VM. In this formula, we utilize the joules consumed

per instruction and the computations processed for acknowledging packet a_i in a VM, given a frequency of f^{opt} and denoted by ${C^{f}}^{opt}_{ai}$.

$$ComputationEnergy = \sum_{i\,=\,0}^{n} \frac{J}{I} C_{a_i}^{f^{opt}} \tag{3.8}$$

The communication energy is the energy spent by a VM for receiving and sending a task T_x and its subtasks. We assume that the communication between the VMs and meta scheduler is insignificant. Equation 3.9 is used to calculate the energy spent to communicate with the nodes involved in the task. Where, T_r is the time to receive an acknowledgement packet, T_s is the time to transmit an acknowledgement packet, and T_w is the waiting time for the meta-scheduler.

$$CommunicationEnergy = \sum_{i\,=\,0}^{n} \frac{J}{I}(T_r + T_s + T_w) \tag{3.9}$$

3.5 Machine Learning Approaches

In this section, we discuss the theoretical approach of the machine learning phase for identifying the best node among the intermediate nodes. To achieve this aim, data is very crucial. So the subsection, data collection describes the tool used to collect raw data and the method to extract the data points from the collected raw data. Therefore, data collection must be followed by data normalization. Some metrics need to be decided upon the characteristics of the results. Afterward, machine learning models must be designed to completely extract the potential of the data to forecast the node value.

3.5.1 Data Collection

Data collection is the key challenge that must be deal with certain accurate methodology. Hence, the concept of the structure of the data is substantial during the data source exploration. In this approach, the proposal is to use a univariate dataset. Since the univariate dataset is simple and imposes less burden on a cloud as well as quicker to train the machine learning model with this dataset.

The tool used to collect data for this approach is WinMTR. It is the graphical user interface version of Matt's TraceRoute. It is a very popular tool available for the Windows platform [8]. WinMTR has the edge over traditional ping and traceroute programs. It integrates the functionalities of ping and traceroute. This way WinMTR can measure the availability of a node as well as hop-by-hop analysis using ICMP packets. There are few advantages to using this tool. Firstly, this tool can capture packet loss and jitter [4]. Another advantage of this tool is that it can continuously update the drop rate of the nodes in the route. This provides a means to observe the nodes for comparatively longer times.

This method collects data by watching a real-world node for two days. They are a workday (Wednesday) and a weekend (Sunday). Each data point represents the sum of received and lost packets across the transmitted packets. Every 60 seconds, three packets are broadcast over a certain node, and the procedure is repeated for 24 hours. The Figure 3.9 illustrates the data collection process. This process repeats for every packet sent through the intermediate node.

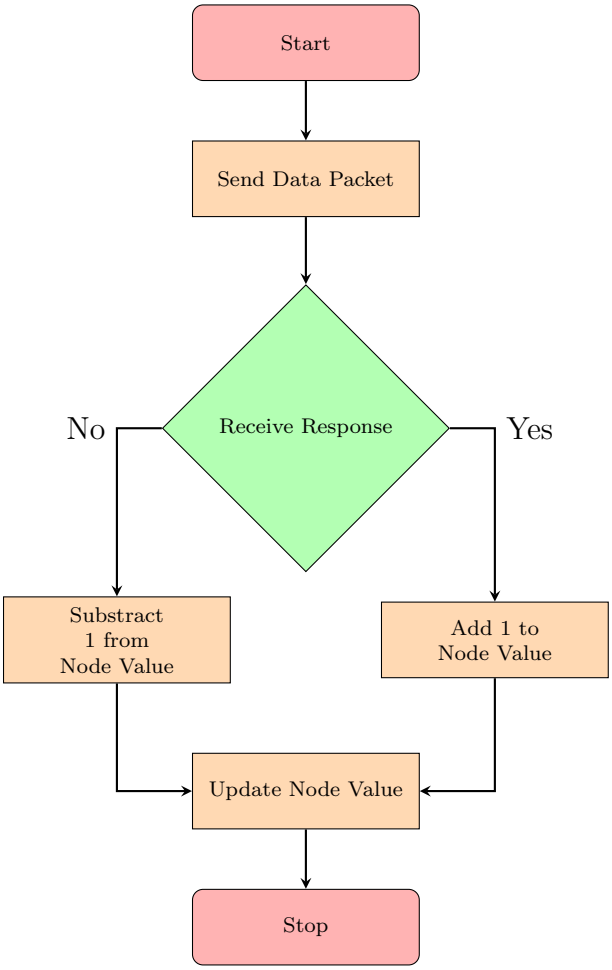


Figure 3.9: Data Collection Process Flow Chart

3.5.2 Data Normalization

Some neural network models are sensitive to the scale of the input data. The difference in the scale of the input variable may increase bias within those neural networks. If the maximum and minimum observation values are greatly different, it makes the neural networks learn large weight values or small weight values. Both of the cases result in inaccurate forecasting because of challenges in generalizing the observations [14].

Data normalization addresses the issues caused by the sensitivity of the observations. Normalization is the process of rescaling all the observations from their original range to the new range between 0 and 1. This step is a part of data preprocessing the data before feeding it to the neural network [17].

There are several types of normalization techniques, among them, the Min-Max normalization technique is the preferred technique for node values dataset. Since the issue in this dataset is the max and min value are too far apart. Using Min-Max, all the observations are rescaled within the range of [0,1]. Equation 3.10 normalizes an observation x to between 0 which is *new min* and 1 which is *new max* with reference to the original scale range [*min*, *max*].

$$y = \frac{(x \text{ - } min) \text{ * } (new\ max \text{ - } new\ min)}{(max \text{ - } min)} + newmin \tag{3.10}$$

Data Splitting

In contrast to the traditional data splitting approach where the dataset is divided into training, validation, and testing subsets, the node values dataset is divided into only training and testing subsets. In time series forecasting there is a strong temporal dependency of the previous observations on the current observations [62]. Therefore it is not fair practice to use conventional validation techniques where the random samples are picked and assigned to either the training set or the testing set. Accordingly, to preserve the temporal dependency between observations, cross-validation on a rolling basis is adopted. Figure 3.10 explains the notion of cross-validation on a rolling basis. It starts with a small portion of data as a training set and another small portion as a test set. The first test is then added to the

training data and the next set of testing values are tested against the later test set. This process repeats for all the test sets.

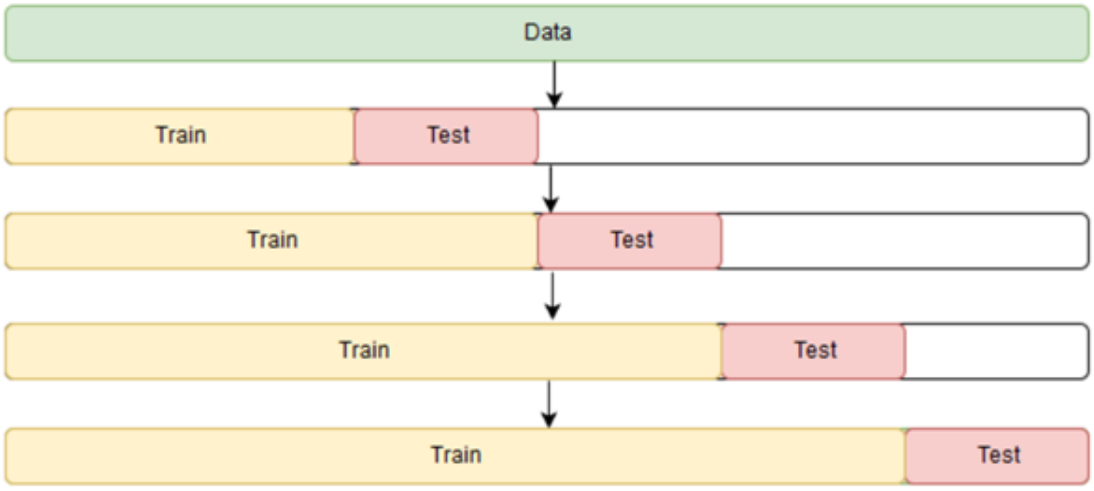


Figure 3.10: Rolling Basis Cross Validation Technique [62]

3.5.3 Metrics Evaluation

Since node value predictions are continuous values, it is a regression problem. Test subset is used to evaluate the model's performance by using symmetric mean absolute percentage error (sMAPE). Alternatively, accuracy is applied to each of the models to evaluate the validity.

sMAPE is an alternative accuracy measure that deals with the limitations of Mean Absolute Percentage Error (MAPE) forecast error measurement. Equation 3.11 generates the sMAPE using forecast value F_t, actual value A_t for n number of observations [58].

$$sMAPE = \sum_{t=1}^{n} \frac{|F_t - A_t|}{(F_t + A_t)/2} \tag{3.11}$$

Accuracy gives the fraction of correct forecasts and the total number of forecasts. Equation 3.12 shows the formula to calculate accuracy using True Positives TP which are correctly forecasted values, False Positives FP which are incorrectly forecasted values, True Negatives

TN which are correctly forecasted values, False Negatives *FN* which are incorrectly forecasted values [42].

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.12}$$

3.5.4 Machine Learning Approaches

Persistence Model

In this research, a persistence model is used as a baseline model. It simply forecasts the node values based on the previous timestep observation. A Persistence model is defined after splitting the train and test subsets. Baseline performance is determined after the model forecasts the node values.

LSTM Model for Node Value Forecasting

LSTM models can be used to define univariate time series forecasting neural networks. Look back concept for sequential data is used to forecast the node values based on the fixed number of previous observations. Look back is simply relying on the number of previous observations to use, to forecast the next observation. Since LSTM models can remember a long sequence of observations, these models are presented with all of the data in batches, timesteps, and input dimensions. They are the inputs for the LSTM model. One input sample is pushed to the model at a time, which consists of an array with $b * t * f$. Where b is the batch size of the input sample and f is the number of input dimensions that are fed to the network and t is the number of time steps as represented in the 3D Figure 3.11. The LSTM model receives one sample input and it attempts to forecast the next time step based on the fed input sample. Look back technique allows the model to look n number of previous observations. Specific values of *b, t, f, and n* are presented in Chapter 4.

Our network consists of a model that is deeply connected with an input layer, output layers, and intermediate layers. The LSTM layer acts as the input layer, it is made up of LSTM neurons. A chain of LSTM neurons receives the past information from the hidden state from the last timestep and new information from the input data. The specific details about the number of LSTM neurons and other layers are discussed in Chapter 4. The final

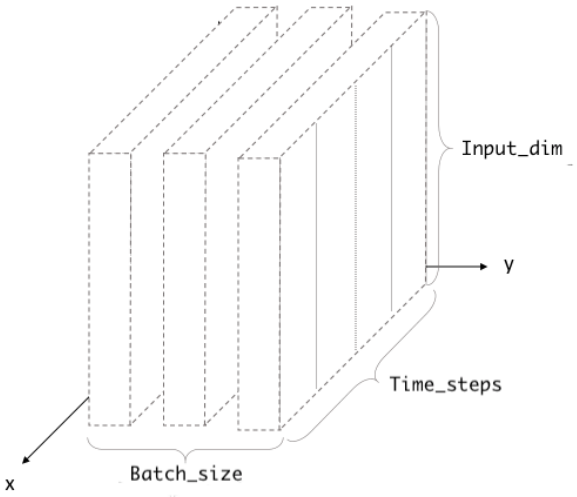


Figure 3.11: Input Dimension Shape of LSTM

layer in the network is an output layer, a dense layer with dimensions of the desired out. Since the node values are sequential, the dense output layer has one neuron. The model is then compiled with a loss function and an optimizer.

N - Beats Model for Node Value Forecasting

N - Beats is another machine learning approach for forecasting node values. It is specially designed to forecast the values of univariate time series. In this technique, the forecasts are delivered based on the backcasts. The model takes node values data up to x_t number of observations as its input and forecast future node value x_{t+h}, where h is the forecast window length. The model learns the sequence of the node values over the look-back period. Look back period is the size of the input $n * H$ from the horizon as shown in the Figure 3.12. The forecast period is H from the horizon.

Specific values of the forecast period and backcast period are mentioned in Chapter 4. The node values are fed to the model along with the index values and series values. If there are multiple groups within the same data, series values are used to identify each group. Since node values are single series, the series column is always constant. The training data is fed as batches to the model with a given network width. Width is decided along with the number

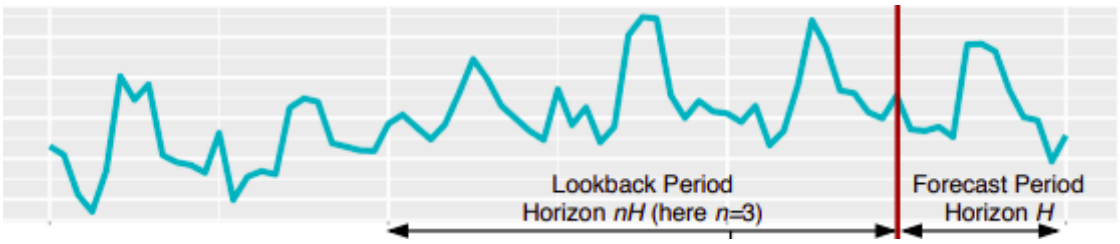


Figure 3.12: N-Beats Time Series in Consideration[53]

of blocks, number of block layers, and prediction length. The width of the network is the widths of the fully connected layers with Relu activation per block.

1. ***Number of Blocks*** - The number of blocks per stack.
2. ***Number of Block Layers*** - The number of fully connected with Relu activation per block.
3. ***Prediction Length*** - It is the length of the prediction. It is also known as 'horizon'.
4. ***Context length*** - It is the length of the look back period.

All these parameters are required to define the N-Beats model. The number of blocks is set to 3 for the interpretable forecast. Similarly, The number of block layers is set to 4. The width of the network is 256 for the trend forecast block and 2048 for the seasonality forecast block.

Chapter 4

Empirical Studies

This chapter investigates the experiments carried out for node value forecasting including data collection, node value data analysis, forecasting model development, and scheduling & energy models simulation and performance evaluation. Data collection introduces the source, tools, and method for collecting the node values. Data preparation is the process of arranging the collected data to feed the machine learning models. Node value data analysis contributes to the statistical analysis of the node values. Predictive model development demonstrates the several model development methods, results, and comparison of the models. Scheduling and energy models simulation and performance evaluation show the execution results of VFA and VTA algorithms.

4.1 Data Collection

The node values are collected over two days by probing a node in a network. The aim of collecting data is to capture the reaction of an intermediate node when a data packet is transmitted through it. Therefore, ICMP packets are sent to the 'spectrum.com' domain every 60 seconds using the WinMTR network monitoring tool. The ICMP packets take 18 identifiable hops to reach the destination since the packets need to pass through one local ISP then through one backbone ISP and then to the destination local ISP. The packet passes through the following list of nodes to reach the destination as shown in the Table 4.1.

Table 4.1: ICMP Packets Route to Destination

Hop	*IPAddress*	*HostName*	*DomainName*
1	192.168.4.1	*LocalComputer*	–
2	192.168.4.1	*LocalComputerPrivateIP*	–
3	24.222.227.149	$nwgl - asr1.eastlink.ca$	*Eastlink*
4	24.215.102.149	$ns - hlfx - dr002.ns.eastlink.ca$	*Eastlink*
5	24.215.102.221	$ns - hlfx - br002.ns.eastlink.ca$	*Eastlink*
6	62.115.147.124	$motl - b2 - link.ip.twelve99.net$	$TELIANET - LIR$
7	62.115.137.142	$nyk - bb1 - link.ip.twelve99.net$	$TELIANET - LIR$
8	62.115.141.244	$rest - bb1 - link.ip.twelve99.net$	$TELIANET - LIR$
9	62.115.123.123	$ash - b2 - link.ip.twelve99.net$	$TELIANET - LIR$
10	62.115.188.211	$svc013937 - ic325643.ip.twelve99.net$	$TELIANET - LIR$
11	209.18.43.58	$209 - 18 - 43 - 58.dfw10.tbone.rr.com$	$CharterCommInc$
12	66.109.5.164	–	$CharterCommInc$
13	66.109.3.1	$ae - 0 - 0.c0.chi75.tbone.rr.com$	$CharterCommInc$
14	66.109.6.249	–	$CharterCommInc$
15	165.237.51.255	$165 - 237 - 51 - 255.twcable.com$	$CharterCommInc$
16	165.237.51.244	$165 - 237 - 51 - 244.twcable.com$	$CharterCommInc$
17	165.237.4.135	$165 - 237 - 4 - 135.twcable.com$	$CharterCommInc$
18	165.237.25.109	$165 - 237 - 25 - 109.twcable.com$	$CharterCommInc$
19	142.136.168.58	*twcdigitalphone.com*	$CharterCommInc$

An intermediate node *nyk-bb1-link.ip.twelve99.net* at hop 7 with IP addresses in the range 62.115.128.0 - 62.115.143.255 is selected to monitor since it shows the maximum tendency to drop packets. Figure 4.1 illustrates the geographical location of the node. Currently, the IP address of the node is 62.115.137.142 which is in the range mentioned before. ICMP packets reach the chosen node at the 7th position from the source node. In Figure 4.1, the cyan color represents the latency of the node. Even though the chosen node does not demonstrate the maximum latency, it is dropping packets in reality.

The additional details about the selected intermediate node are mentioned below:

- **NetRange** : 62.115.128.0 - 62.115.143.255
- **CIDR** : *not provided*
- **NetName** : EU-TELIANET-20130924

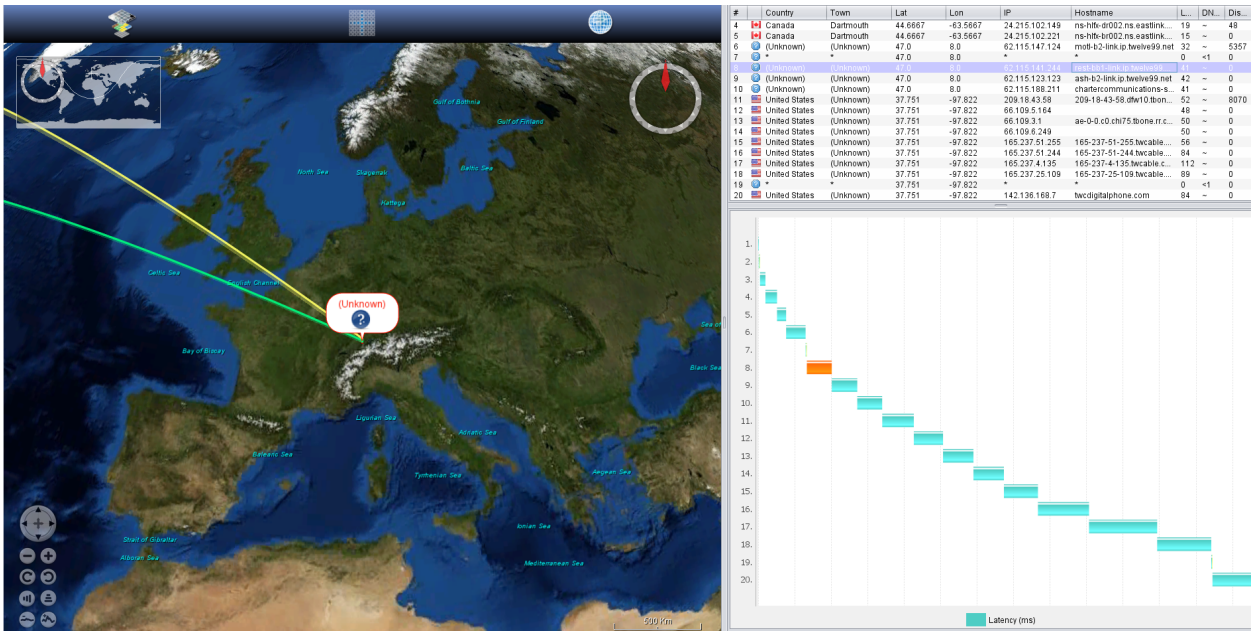


Figure 4.1: Geographical Location of the Selected Node

- **NetHandle** : TELIANET-LIR
- **Parent** : 62.115.0.0 - 62.115.255.255
- **Origin** : AS1299
- **Organization** : Telia Company AB
- **Created** : 2004-04-17
- **Updated** : 2020-12-16
- **Address** : 16979 Solna
- **State and Country**: Sweden

If the node transmits the sent ICMP packet, it replies with a response packet. The response packet is collected by the local computer and updates the sent count and received count for that node. Similarly, if the sent ICMP packets are dropped, the chosen node fails to send a response packet. This action triggers the sender to update the sent count but the

received count remains constant. This process repeats for every ICMP packet sent to the chosen node.

4.2 Data Preparation

In this step, the raw data collected about the chosen node needs to be transformed as univariate time series data. Therefore, the node value of a particular node initiates at zero and a reward of +1 is awarded to its node value if a response is received for a sent ICMP packet. Accordingly, a reward value of -1 is awarded when the node fails to respond to the sent ICMP packet. This process is applied to the data collected during a weekday and a weekend. Figure 4.2 illustrates the 5012 data points recorded on a weekday and 4.3 illustrates the 5209 node values observed on a weekend, from the telia intermediate node. Since the drop rate of this node is around 77.85% on the weekday and 78.96% on the weekend, a gradual dip is observed on both graphs.

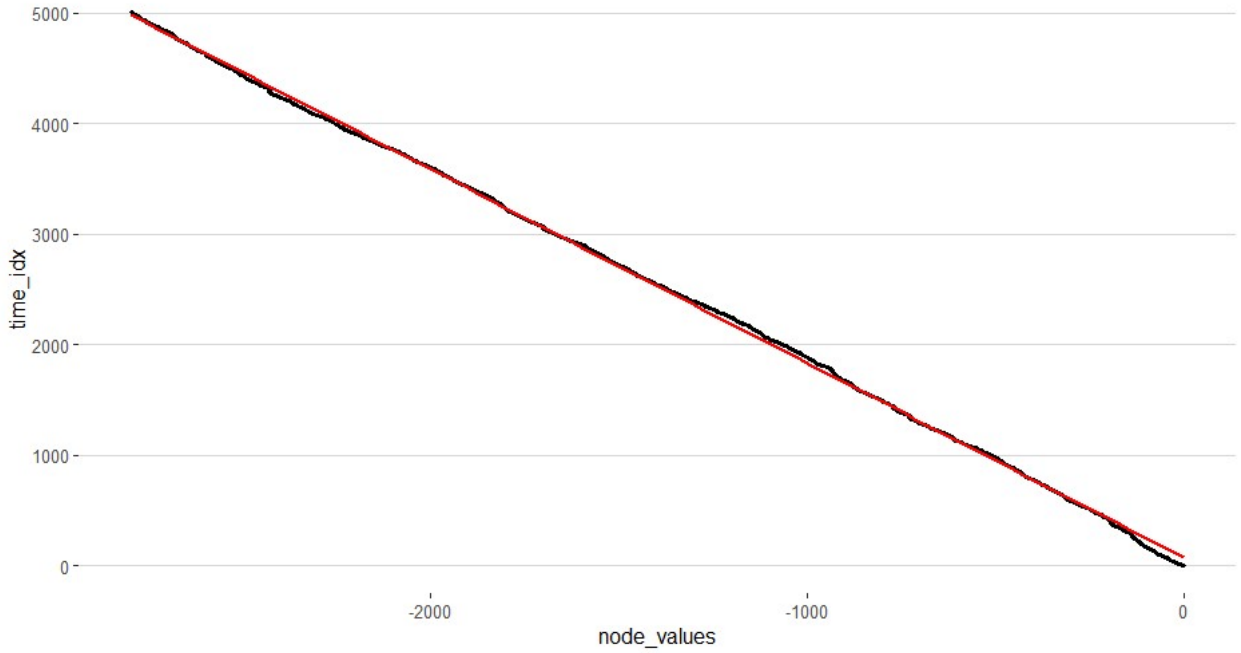


Figure 4.2: Last 100 Node Values in a Weekday

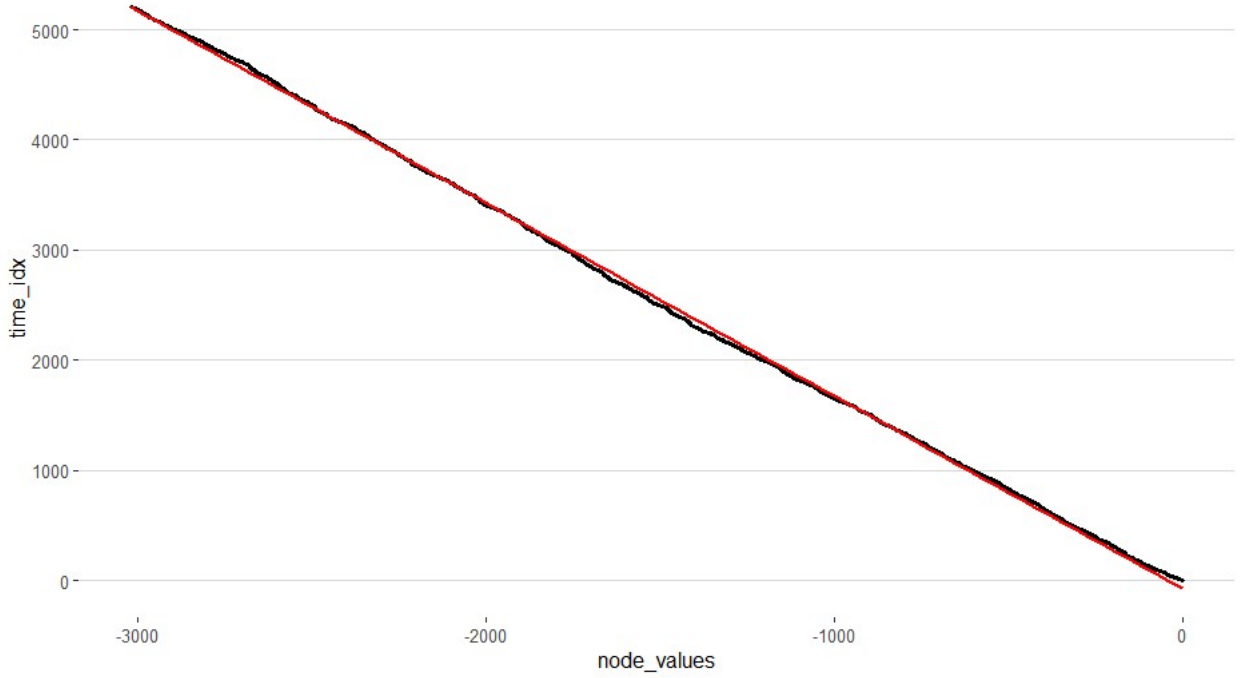


Figure 4.3: Last 100 Node Values in a Weekend

Graph of all the node values is very concentrated, therefore, a candle graph is prepared as illustrated in the Figure 4.4 for weekday node values and Figure 4.5 for weekend node values to observe the individual transmission outcomes. They are the graphs of the last 100 transmission results observed on a weekday and a weekend. The grey indicators show the dropped packets and the cyan indicators show the successfully transmitted packets. There is almost a pattern repeating over every few data points along with some noise.

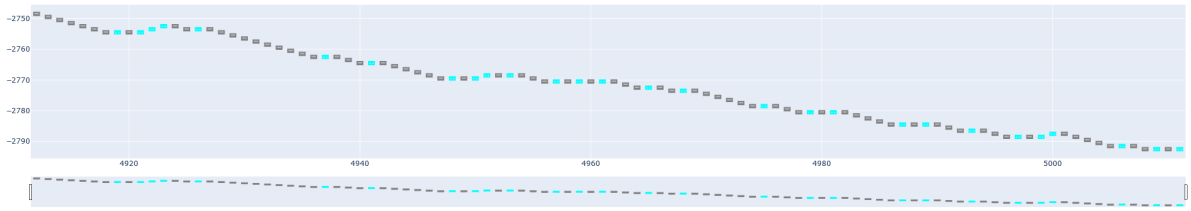


Figure 4.4: Last 100 Node Values in a Weekday

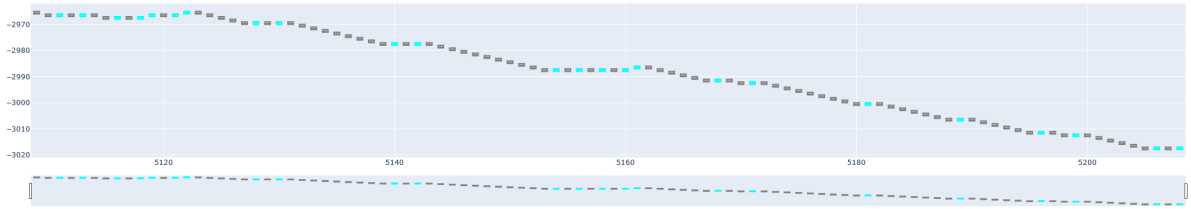
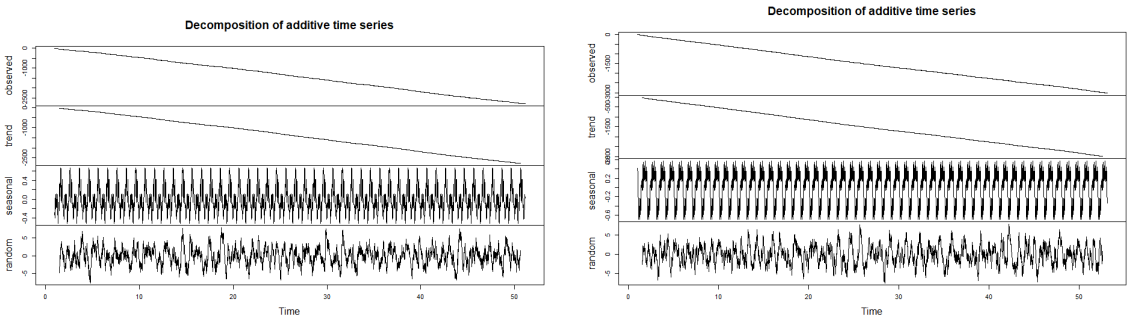


Figure 4.5: Last 100 Node Values in a Weekend

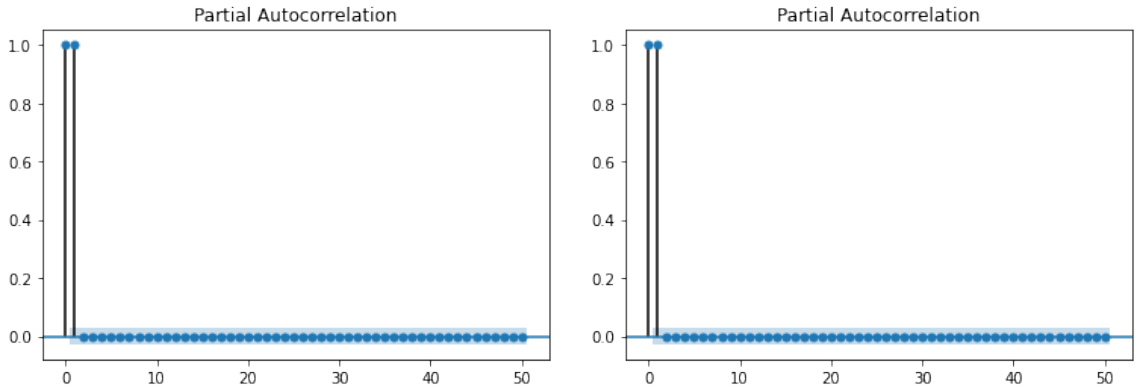
To understand the decomposition of the weekday data, a decomposition graph is prepared. Figure 4.6a illustrates the observed, trend, seasonality, and noise graphs overtime on a weekday. Similarly, Figure 4.6b represents the time series decomposition of node values on a weekend. The frequency for this model is set to 100 for both the plot, which means the seasonal pattern repeats after every 100 transmissions. This is an additive decomposition model with time on the x-axis and four decomposition graphs on the y-axis. It can be seen that the trend seems obvious from the observed graph. The seasonal graph extracted from the node values seems plausible and it is showing a pattern over time. The noise graph is displaying interesting aspects of the data like repeating high variability over a period of time.



(a) Decomposition of Weekday Time Series Data (b) Decomposition of Weekend Time Series Data

Figure 4.6: Node Values Time Series Decomposition

The following Figure 4.7a displays the correlation of node values of weekday time series and Figure 4.7b shows the correlation of node values of weekend time series. The lag is set to 50 to construct both the PACF plots. The sole purpose of the PACF plots in this research is to help identify the number of node values to look back before making a forecast with the LSTM model. In this scenario, the data is showing a strong correlation between current node values and the last two node values on weekday time series data as well as on weekend time series data.



(a) PACF Plot of Weekday Time Series Data (b) PACF Plot of Weekday Time Series Data

Figure 4.7: Observed Partial Autocorrelation of Time Series

4.3 Forecasting Model Development

4.3.1 Objective

This research aims to investigate the possibility of predicting the packet drops using the transmission history of a node during a weekday and a weekend. Persistence model as a baseline model, LSTM and N-Beats models as deep learning models are compared with each other in terms of node value forecast accuracy and node transmission outcome accuracy.

4.3.2 Forecasting Models

Persistence Model

To determine the baseline model performance of the deep network models, a persistence model is implemented using the last 1000 node values as the test dataset and the rest of the node values are utilized as a training dataset. To simplify the training with univariate data, the node values are transformed into a supervised learning problem. A lagged representation is generated to predict the observation at t given the observation at t-1. The persistence model is defined as a function that returns the value sent in as input. Because no model training or retraining is necessary, we just go through the test dataset time by time step and receive predictions. Once predictions for each time step in the training dataset are forecasted, they are compared to the true test values, and a SMAPE score and MAE score is calculated.

LSTM Model

After cleaning the data, normalizing the data is important since the LSTMs are sensitive to the scale of the input data. It is a good idea to rescale the data to the 0 to 1 range, commonly known as normalizing. Using the scikit−learn library's MinMaxScaler preprocessing class, the dataset is simply normalized.

As mentioned in section 3.5.4, an LSTM layer needs a three−dimensional input and LSTMs will generate a two−dimensional output as an evaluation from the conclusion of the

sequence by default. We can define an LSTM model for node value forecasting problem as shown in the Figure 4.8.

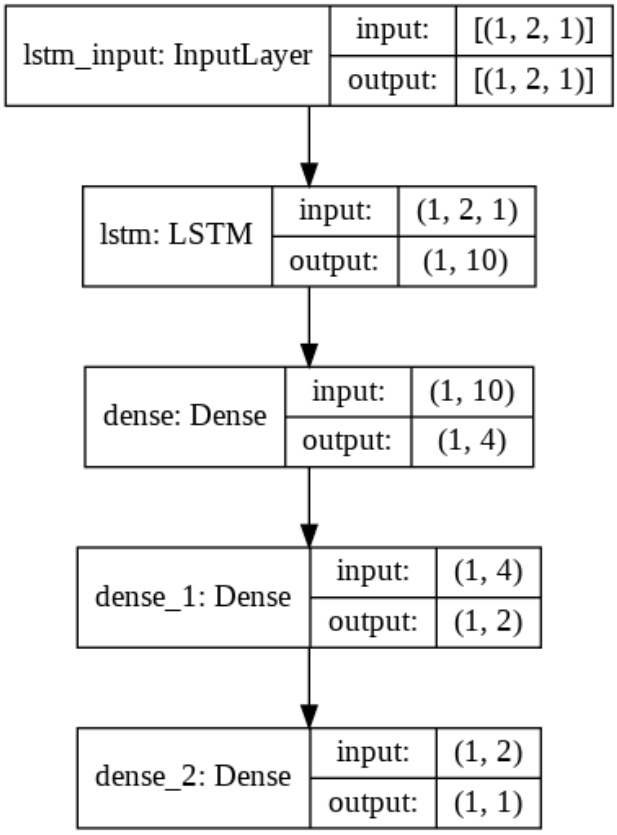


Figure 4.8: LSTM Neural Network Architecture for Forecasting Packet Droppings

The shape of the input is critical in the specification; this is what the model expects as input for each sample in terms of the number of time steps and the number of characteristics. We're dealing with a univariate series, thus there is just one feature for one variable. This model accepts 1 sample at a time and looks back at 2 timesteps since the PACF Figure 4.7a shows that the last two node values impact the current node value. Since the node value data is univariate, there is only one feature. After several experiments, the LSTM model is developed with an input LSTM layer, 2 dense layers, and an output dense layer in the end. In this problem, we build a model with 10 LSTM units in the hidden layer and the following dense−dense hidden layers are stacked with 10 and 4 units respectively. A single numerical value is predicted by the output layer, therefore, the final output layer has 1 dense unit.

After fitting the node value data into the model and estimating the model's performance on the training dataset, the model's competence on new and unseen data is tested using a test dataset. Normally, this is accomplished with cross-validation for a typical classification or regression task. The sequence of node values is crucial when working with our time series data. Therefore, nested cross-validation is carried out using 4 subsets of data. Split 1, split 2 and split 3 are three different models that contain different sizes of training and testing subsets. The following table 4.2 refers to the data split information. The first model is trained with around 2000 node values and is tested with the next 1000 node values. The second model's training is the combination of train and test subsets of the first model. Similarly, the third model's training data is the merger of the second model's training and testing subsets. All the three model's accuracy is calculated by the average of the three model's results.

Table 4.2: LSTM Nested Cross-validation Split Information

	$Split1$	$Split2$	$Split4$
$WeekdayTraining$	$1-2009$	$1-3010$	$1-4011$
$WeekdayTesting$	$2010-3010$	$3011-4011$	$4012-5012$
$WeekendTraining$	$1-2206$	$1-3207$	$1-4208$
$WeekendTesting$	$2207-3207$	$3208-4208$	$4209-5209$

.

N-Beats Model

The data must be separated into training and validation time series datasets before modeling. Because they are specifically intended to handle time series data, Pytorch dataloaders are utilized for this job. The data is then converted to torch tensors to hold the information about static and time−varying variables. Data loaders take additional parameters like max_encoder_length to to specify the time series dataset's maximum history length and max_prediction_length to set the maximum forecast length. Later, hyperparameters are set for the model. The widths are the most important hyperparameters in the NBeats model. Each represents the breadth of a forecasting block. The first anticipates the trend by default, whereas the second forecasts seasonality. The following table 4.3 describes the best hyperparameters for node values dataset. *backcast_loss_ratio* is the weight of backcast in comparison to forecast when calculating the loss. A weight of 1.0 means that forecast and

backcast loss is weighted the same. *Context_length* is, number of time units that are used to condition the forecasts. Also referred to as the 'lookback period'. It should be between 1 and 10 times the length of the prediction. *sharing* refers to whether the weights are shared with the other blocks per stack. *widths* denotes the widths of the fully connected layers with ReLu activation in the blocks.*num_block_layers* is the number of fully connected layers with ReLu activation per block. *num_blocks* is the number of blocks per stack. Since N−Beats is a dual stack architecture, this model has two stacks.

Table 4.3: N-Beats Hyperparameter Configurations

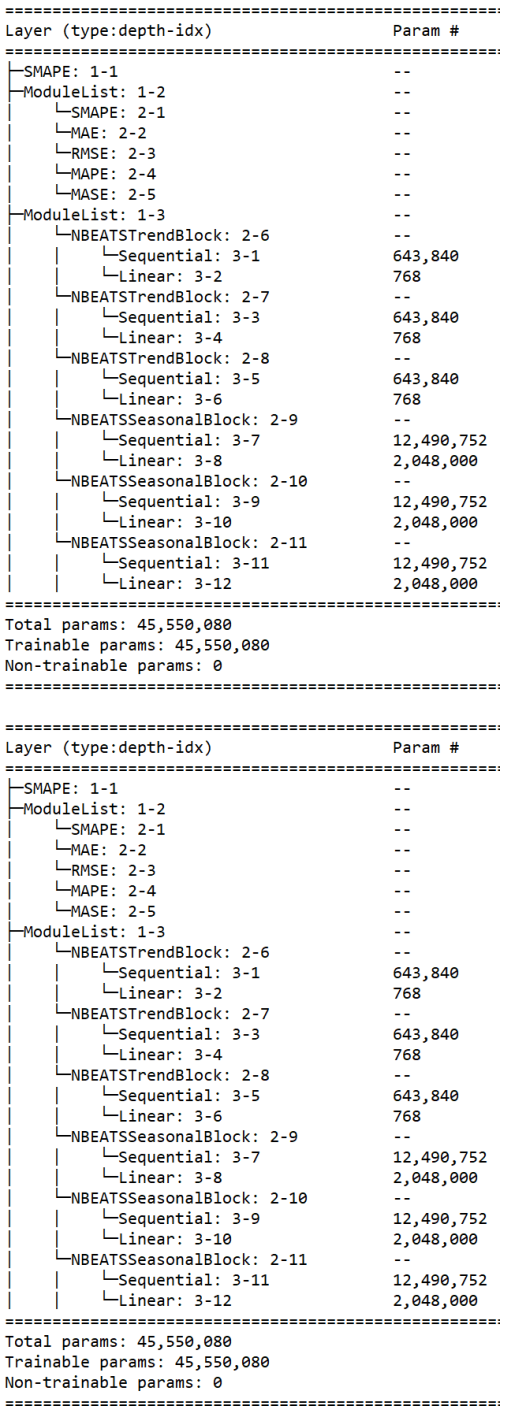
$Hyperparameter$	$Value$
$backcast\_loss\_ratio$	1.0
$context\_length$	2000
$dropout$	0.1
$expansion\_coefficient\_lengths$	$[3, 7]$
$learning\_rate$	0.005
num_block_layers	$[3, 3]$
num_blocks	$[3, 3]$
$prediction\_length$	1000
$sharing$	$[True, True]$
$stack\_types$	$['trend', 'seasonality']$
$widths$	$[256, 2048]$

.

When training deep neural network topologies, N-BEATS largely depends on the well-established notion that transmitting residual changes of the input signal via stacks of layers gives clear advantages. The authors of N-BEATS propose the concept of backward predictions, often known as "backcasts," in which the model attempts to recreate the time series that it received as an input. Backcast residuals are removed from the output of the previous fully-connected block (or the input signal in the case of the first block) and transferred to the next block. The forecasts, a second branch of block outputs, are saved and totalled to create the final predictions.

As illustrated in Figure 4.9, the researcher designed the N−Beats model with dual stack approach for the node value predicting issue. Both the stacks are like a pipeline and the first stack is very much similar to the second stack. In each stack, there are three consecutive trend blocks followed by three consecutive seasonal blocks. Each trend of the block is completely

linked and consists of a linear layer that accepts 2000 node values and produces 256 values. It is then followed by a layer that has been activated using ReLu. It is followed by another sequential layer that contains a dropout layer and a linear layer that intakes 256 values and outputs 256 values. There is another ReLu triggered layer after the sequential layer. After ReLu, one more sequence of dropout layer and linear layer. The block concludes with a ReLu layer following the sequential layer. Based on 256 inputs, each block creates a linear prediction θ_l^f and a linear backcast θ_l^b, each of which yields three values. The seasonal block's architecture is similar to the trend block's, but the number of inputs and outputs of the linear layers differs. The initial linear layers of the seasonal block take in 2000 values and produce 2048 values. Whereas, intermediate linear layers take in 2048 values producing 2048 values. At last, the final linear forecast layer θ_l^b and backcast layer θ_l^b takes in 2048 values producing the final output of the last 1000 values as forecast values.



```
==========================================================
Layer (type:depth-idx)                   Param #
==========================================================
├─SMAPE: 1-1                            --
├─ModuleList: 1-2                       --
│    └─SMAPE: 2-1                       --
│    └─MAE: 2-2                         --
│    └─RMSE: 2-3                        --
│    └─MAPE: 2-4                        --
│    └─MASE: 2-5                        --
├─ModuleList: 1-3                       --
│    └─NBEATSTrendBlock: 2-6            --
│    │    └─Sequential: 3-1             643,840
│    │    └─Linear: 3-2                 768
│    └─NBEATSTrendBlock: 2-7            --
│    │    └─Sequential: 3-3             643,840
│    │    └─Linear: 3-4                 768
│    └─NBEATSTrendBlock: 2-8            --
│    │    └─Sequential: 3-5             643,840
│    │    └─Linear: 3-6                 768
│    └─NBEATSSeasonalBlock: 2-9         --
│    │    └─Sequential: 3-7             12,490,752
│    │    └─Linear: 3-8                 2,048,000
│    └─NBEATSSeasonalBlock: 2-10        --
│    │    └─Sequential: 3-9             12,490,752
│    │    └─Linear: 3-10                2,048,000
│    └─NBEATSSeasonalBlock: 2-11        --
│    │    └─Sequential: 3-11            12,490,752
│    │    └─Linear: 3-12                2,048,000
==========================================================
Total params: 45,550,080
Trainable params: 45,550,080
Non-trainable params: 0
==========================================================

==========================================================
Layer (type:depth-idx)                   Param #
==========================================================
├─SMAPE: 1-1                            --
├─ModuleList: 1-2                       --
│    └─SMAPE: 2-1                       --
│    └─MAE: 2-2                         --
│    └─RMSE: 2-3                        --
│    └─MAPE: 2-4                        --
│    └─MASE: 2-5                        --
├─ModuleList: 1-3                       --
│    └─NBEATSTrendBlock: 2-6            --
│    │    └─Sequential: 3-1             643,840
│    │    └─Linear: 3-2                 768
│    └─NBEATSTrendBlock: 2-7            --
│    │    └─Sequential: 3-3             643,840
│    │    └─Linear: 3-4                 768
│    └─NBEATSTrendBlock: 2-8            --
│    │    └─Sequential: 3-5             643,840
│    │    └─Linear: 3-6                 768
│    └─NBEATSSeasonalBlock: 2-9         --
│    │    └─Sequential: 3-7             12,490,752
│    │    └─Linear: 3-8                 2,048,000
│    └─NBEATSSeasonalBlock: 2-10        --
│    │    └─Sequential: 3-9             12,490,752
│    │    └─Linear: 3-10                2,048,000
│    └─NBEATSSeasonalBlock: 2-11        --
│    │    └─Sequential: 3-11            12,490,752
│    │    └─Linear: 3-12                2,048,000
==========================================================
Total params: 45,550,080
Trainable params: 45,550,080
Non-trainable params: 0
==========================================================
```

Figure 4.9: N−Beats Neural Network Architecture for Forecasting Packet Droppings

4.4 Results

Metrics used to evaluate the models are SMAPE and MAE. Two of the scores are reported along with the packet dropping prediction accuracy for each of the models, LSTM and N-Beats.

4.4.1 Persistence Model Results

Finally, the Figure 4.10 illustrates the subgraphs with 250 forecasted node values against actual node values sequentially from the expected node values in the weekday test dataset. The plots of Persistence models show that the model is one step behind the actual values. In the forecast statistics, there is a downward trend but the packet dropping prediction accuracy of the model is 58.7%, which highlights the limits of the Persistence model. As expected, the MAE of the model is 1, since the current node value follows the previous node value. SMAPE is calculated as 0.040, this value holds minimal significance to determine the accuracy of the model in this case.

Furthermore, the weekend test dataset is evaluated with a similar persistence model. The following Figure 4.11 shows the precision of the persistence model in four subgraphs. Each of the subgraphs displays the 250 actual node values and forecasted node values in a sequence. Similar to the weekday result, the MAE of the model is 1 because of the persistence model's forecasting approach. The forecast error of the weekend is measured as 0.037 SMAPE, which is slightly below the weekday's SMAPE. The accuracy of the packet dropping prediction of the weekend model is comparatively better than a weekday, which is about 62.1 %.

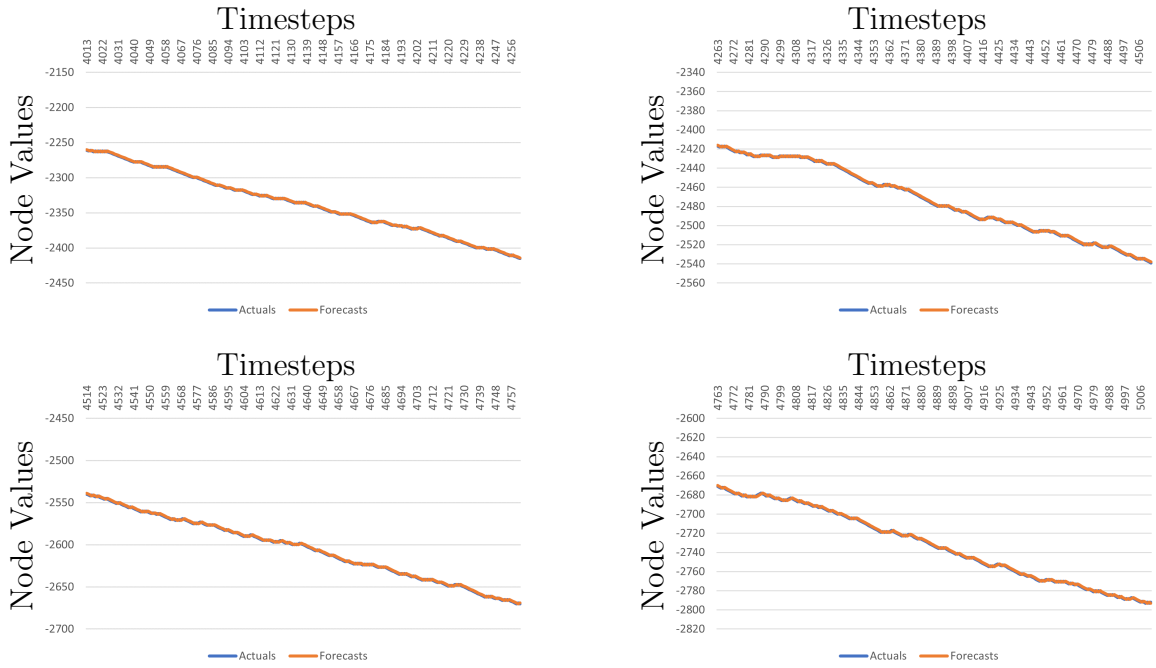


Figure 4.10: Graphs of the Actuals vs Forecasts produced by Persistence Algorithm on a Weekday

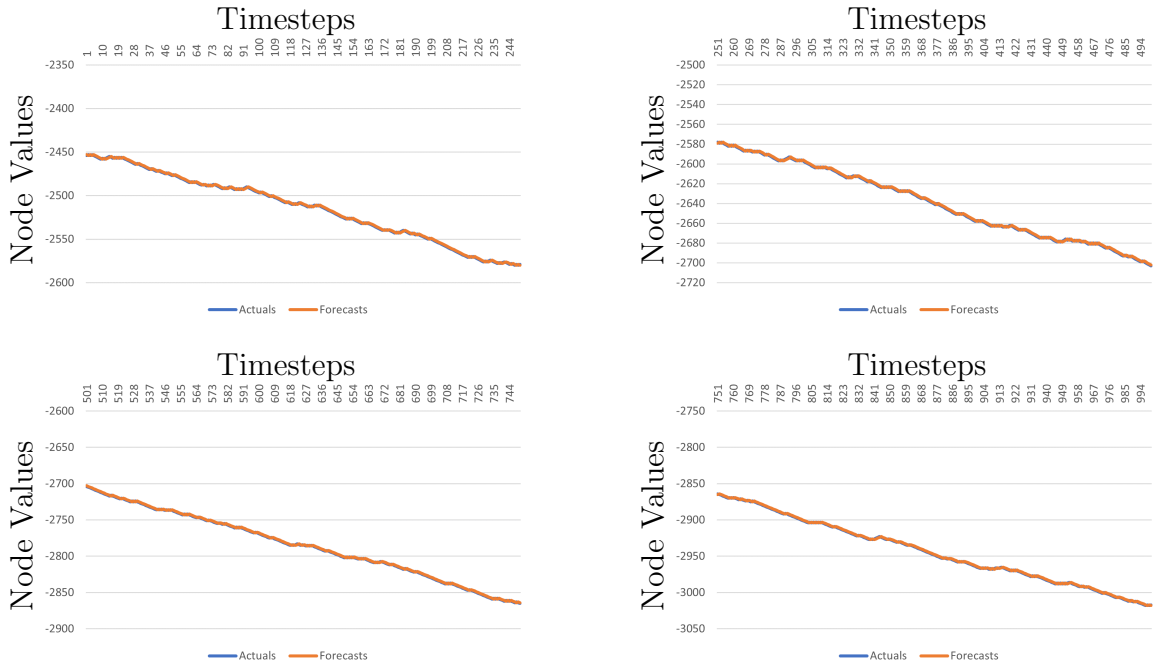


Figure 4.11: Graphs of the Actuals vs Forecasts produced by Persistence Algorithm on a Weekend

4.4.2 LSTM Model Results

The individual results of each split of the weekday model and weekend model are discussed in the following parts.

First Split

The initial nested Cross-validation set consists of 2009 node values for training and 1000 testing node values on the weekday. The weekend training set is a bit larger with 2206 node values and the model is tested on 1000 node values. The figures 4.12 and 4.13 displays the forecasted node values versus the actual node values for the first split. On both the weekday and weekend, the first subgraph displayed a large discrepancy in the forecasts. However, the forecasts recovered sharply and followed the actual node values very closely. The second subgraph shows that the forecast node values are closely following the actual node values across the timesteps on both graphs. The third subgraphs of each of the first split plots display close to forecast and actual node values but there is a gentle rise in the gap between forecast and actual node values. The final subgraph of the weekday plot shows the forecast and actual node values are slightly closer than the forecast and actual values of the final subgraph of the weekend plot.

The errors SMAPE and MAE for the weekday are calculated as 0.251 and 4 respectively. For the same weekday, the first split's accuracy is at 80.4%. In the same way, the SMAPE and MAE for the first split of the weekend model are measured as 0.311 and 5 respectively. As a result, the accuracy of the forecast values of the weekend's first split model is measured at 77.3%.

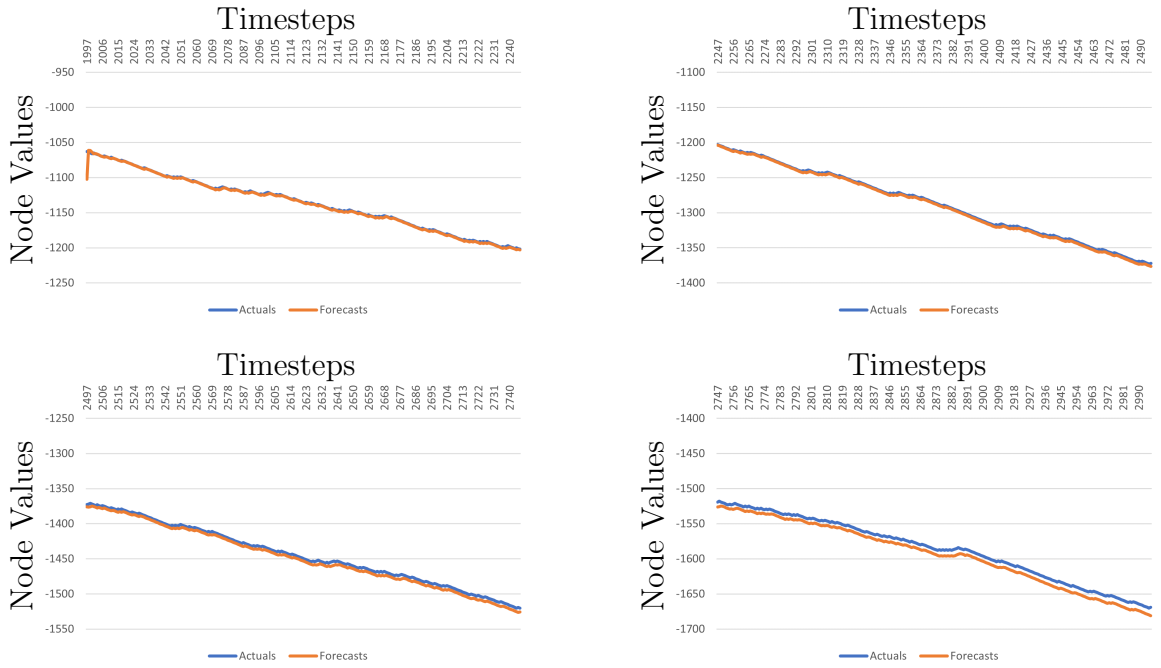


Figure 4.12: Graphs of the Actuals vs Forecasts produced by First Split LSTM Model on a Weekday

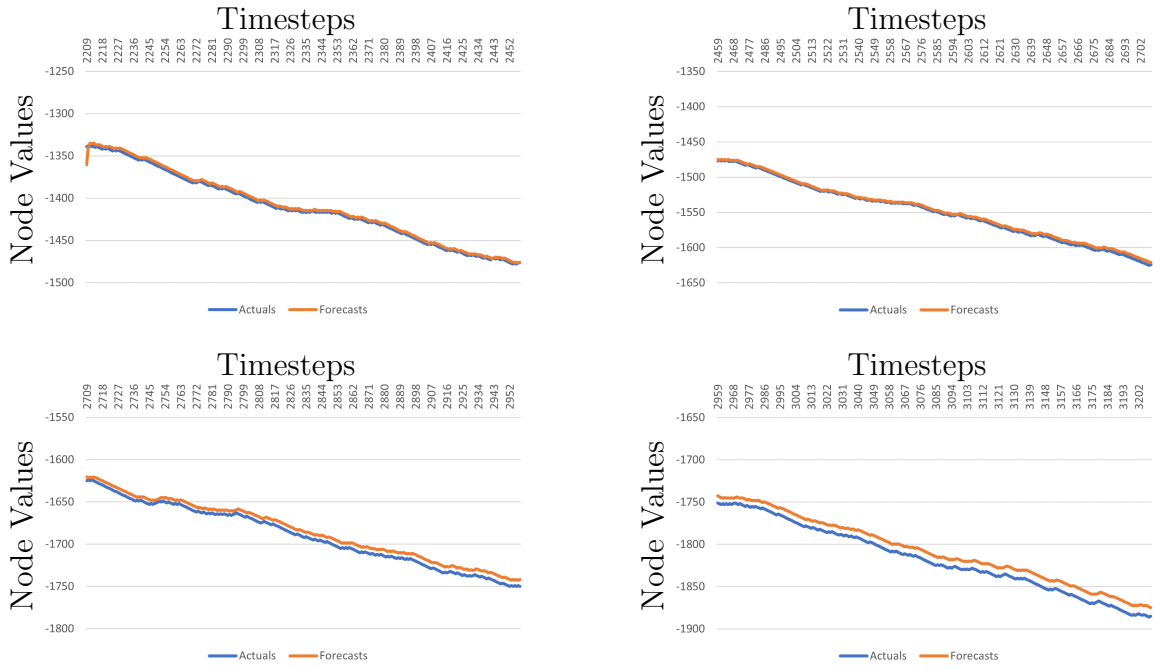


Figure 4.13: Graphs of the Actuals vs Forecasts produced by First Split LSTM Model on a Weekend

Second Split

The figures 4.14 and 4.15 illustrate the forecasting results of the weekday model and weekend model. This second split is part of the nested cross-validation process to validate the LSTM model. Its training dataset is the combination of the node values used in training and testing subsets of the first split. The weekday subgraphs show interesting results. The first graph begins with poor forecasts, however, the forecast node values quickly close the gap and continue to bring the node values towards the actual values. This process continues into the second subgraph where the LSTM model caught most of the turbulence in the node values. In the third subgraph, the forecast node values begin converging towards the actual values and almost touch the actual node values trendline. In the final subgraph of the weekday results, the forecast node values and actual node values adjoin the actual node values and remain merged for most of the remaining timesteps. Although the trendlines of actual and forecast node values run very close, the errors SMAPE and MAE are measured as 0.306 and 6 respectively. The accuracy of the obtained forecasted node values is 79.4%.

The weekend's first subgraph, like the first subgraph of the workday, is generated by significantly erroneous forecast node values. The model rapidly followed up to the real node values and stayed that way for the rest of the subgraph. The second subgraph is also displaying that the forecast node values are running very close to the actual node values and continues to be closed for the rest of the graph. The second subgraph likewise shows that the forecast node values are extremely close to the actual node values and will remain so for the remainder of the graph. Similar to the second subgraph, the third subgraph also shows that the forecast node values are maintained relative to the actual node values for the eternity of the graph. The last subplot illustrates the weekend model's outstanding forecasting performance by forecasting node values that are significantly closer to the actual node values. This model has the lowest SMAPE and MAE of 0.111 and 2, respectively. The accuracy of packet dropping prediction is 62.4 %.

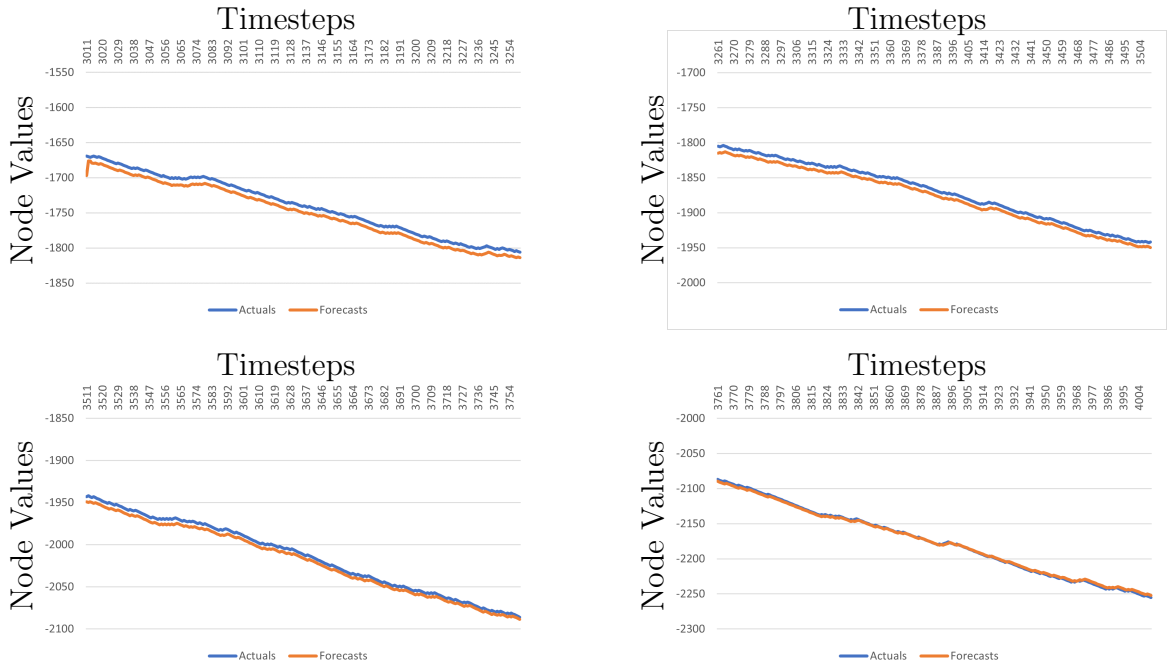


Figure 4.14: Graphs of the Actuals vs Forecasts produced by Second Split LSTM Model on a Weekday

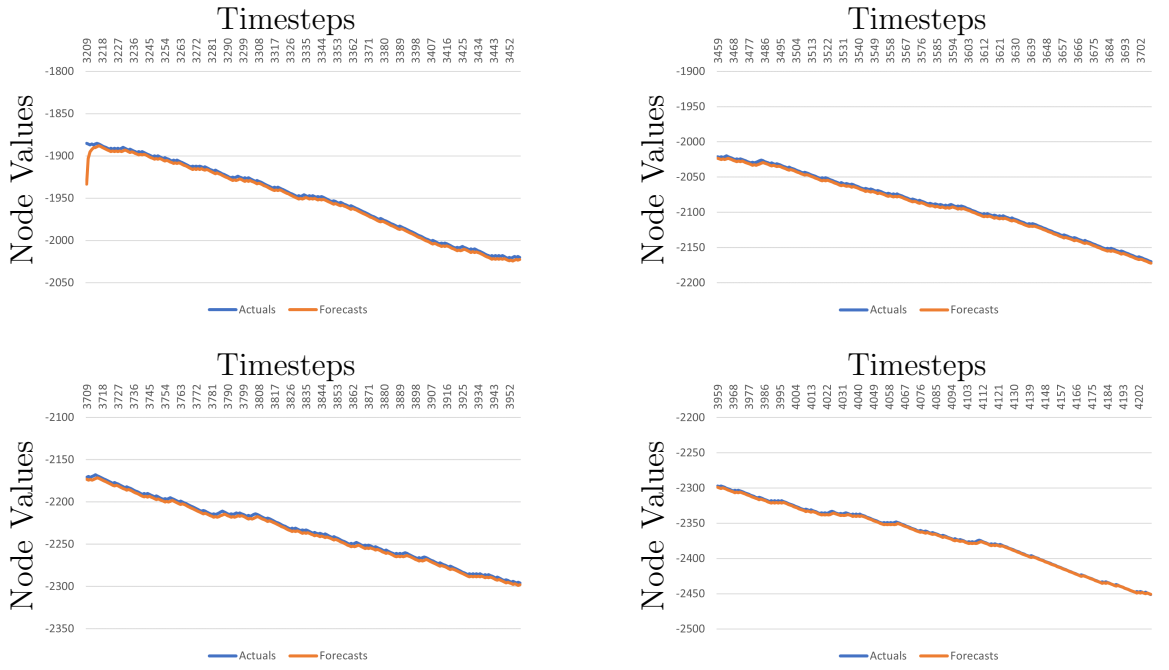


Figure 4.15: Graphs of the Actuals vs Forecasts produced by Second Split LSTM Model on a Weekend

Third Split

Finally, the third split is the final stage of this LSTM model's nested cross-validation. The third split's training dataset is created by stacking the second split's training and testing datasets. The testing dataset is made up of the remaining 1000 node values from the weekday and weekend node value datasets. The results are split into four subgraphs after the model is trained using the training dataset and evaluated with the testing dataset, as illustrated in the Figure 4.16. The first subgraph shows that the difference between forecasts and actuals narrowed quickly, and the graph continues with a tiny gap between forecasts and actuals for the remainder of the node values. In terms of the difference between forecasts and actuals, the second subgraph is quite similar to the third subgraph. For the majority of the graphs in both subgraphs, the forecast node values trendline is parallel to the actual node values. The forecasts and actuals are quite near in the last subgraph, although there is usually a gap between them. The accuracy of the result in terms of packet dropping prediction is determined as 76.8%. Using the same result, the errors SMAPE and MAE of the model are 0.348 and 9, respectively.

For nested cross-validation, the weekend node values are divided into three groups. The latest 1000 node values are stored in the final split's testing set. Following model training, the model's efficiency is assessed by utilizing the unused 1000 node values. When compared to the weekday third split outcomes, the weekend model's subgraphs show a significant divergence in all four of them. The first subgraph shows that the model attempts to close the gap between forecasts and actuals. However, the gap is still wide from the actual node values trendline. The second subgraph shows that the predicted and actual values are parallel with a substantial difference between them. As illustrated in the third subgraph, the distance steadily widens. The final subgraph demonstrates that the model is unable to keep up with the actual node values but is nearly able to comprehend the variations in the outcome. This model accomplished only 62.2% packet dropping prediction accuracy with errors SMAPE and MAE as 1.011 and 28, respectively.

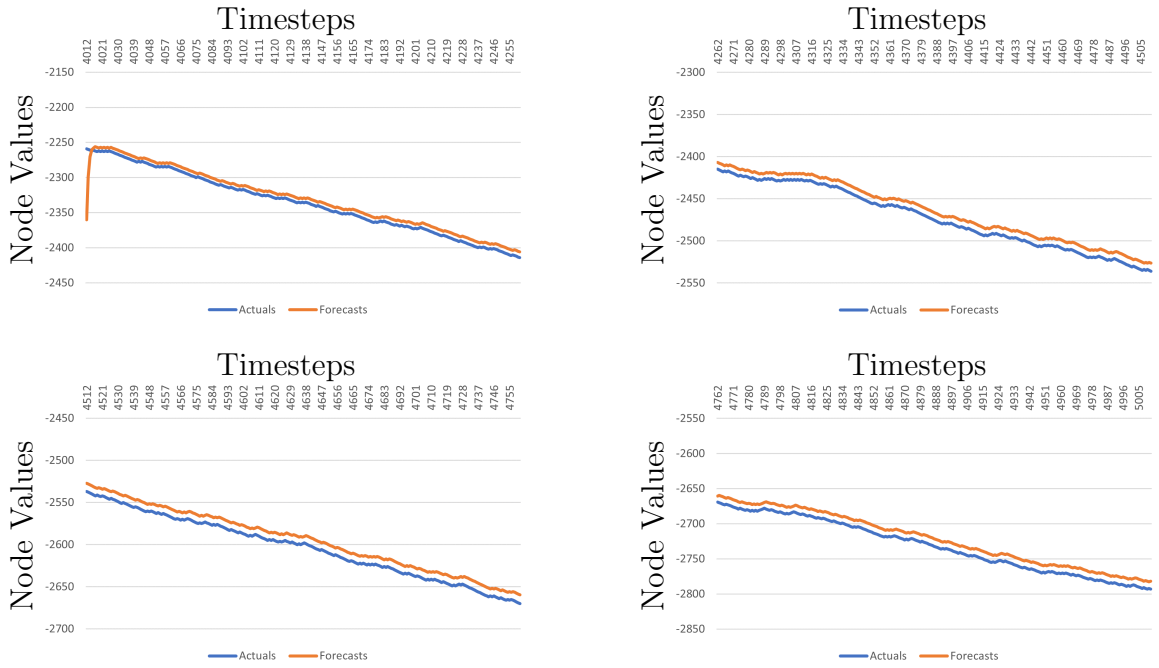


Figure 4.16: Graphs of the Actuals vs Forecasts produced by Third Split LSTM Model on a Weekday

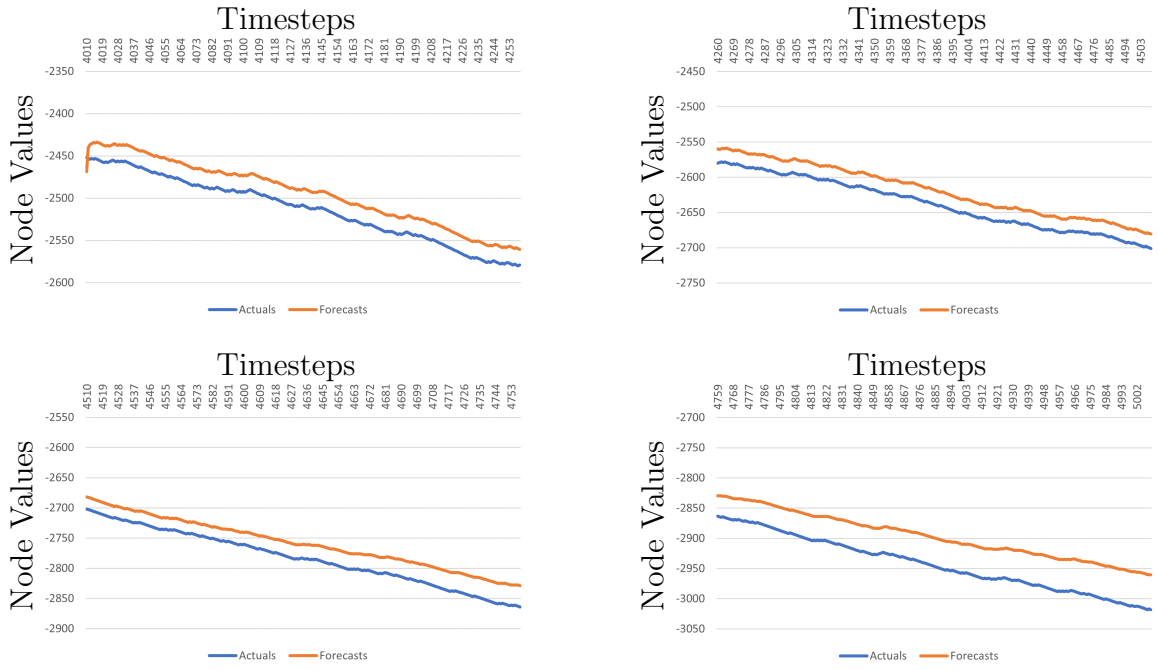


Figure 4.17: Graphs of the Actuals vs Forecasts produced by Third Split LSTM Model on a Weekend

4.4.3 N-Beats Model Results

N-Beats architecture described in section 4.3.2 is tested and analyzed on the last 1000 node values for the weekday node values dataset. The Figure 4.18 presents the weekday node value forecasts against actuals in four subgraphs. Each of the subgraphs represents the 250 values in ascending order. The first subgraph shows the first 250 values of the result. Even though there is a significant oscillation in the forecasts, the forecasts are closely following the actuals and the gap is very minimal. However, the growth in the gap can be observed in the second subgraph that shows the next 250 forecast values against actuals. The gap is increased significantly between the forecasts and the actuals as shown in the third subgraph. In the final subgraph, the forecasts have completely diverged from the actuals. The accuracy of the weekday model in terms of packet dropping prediction is standing at 62.3%. The errors SMAPE and MAE are measured as 0.811 and 21, respectively.

In a similar way, The weekend node value dataset is evaluated using the last 1000 node values. The results are illustrated in the Figure 4.19 as four subgraphs in the increasing order. The first subgraph resembles the first subgraph in the Figure 4.19, the forecasts closely follow the actuals. Unlike the second subgraph in Figure 4.19, the gap between forecasts and actuals remains minimal until the third subgraph. In this third subgraph, the divergence can be observed from the middle section. The gap intensified as the timesteps increased, as shown in the final subgraph of Figure 4.19. statistical measures reveal that the SMAPE and MAE are 0.730 and 21. The precision of the packet dropping prediction is 56.5%.

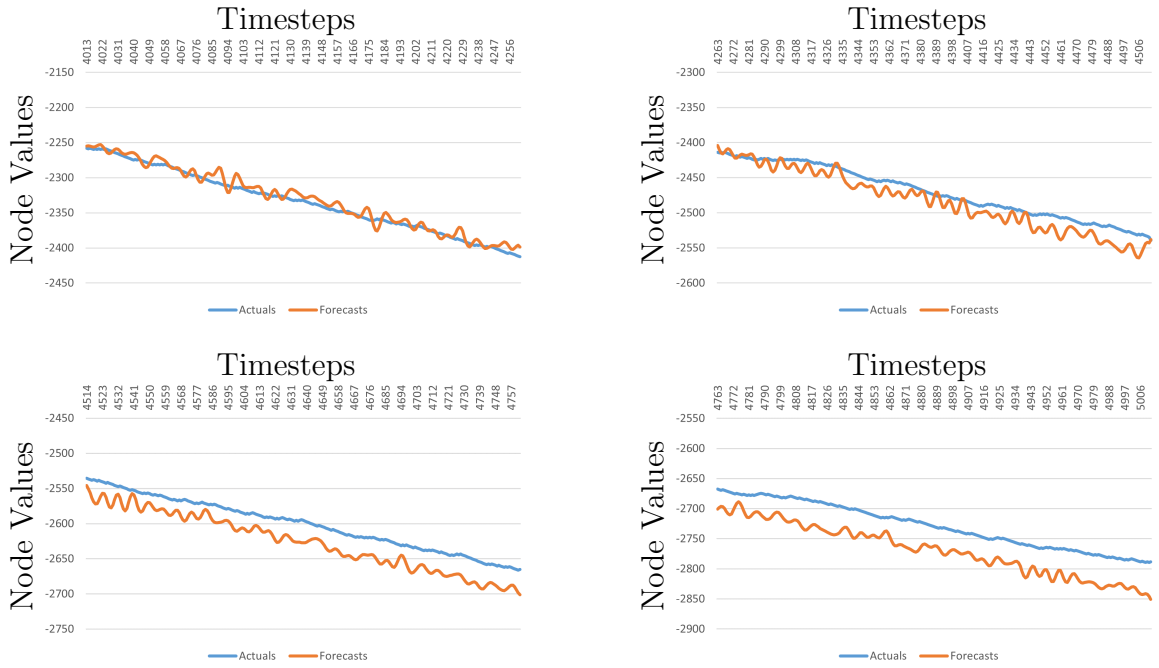


Figure 4.18: Graphs of the Actuals vs Forecasts produced by N-Beats Algorithm on a Weekday

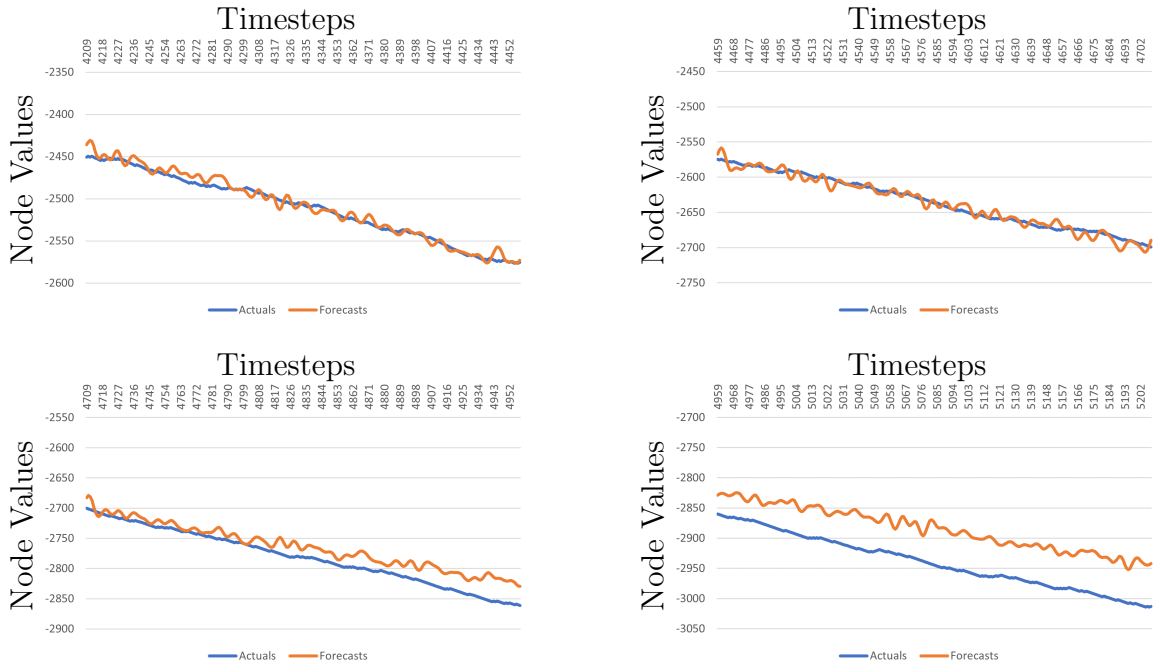


Figure 4.19: Graphs of the Actuals vs Forecasts produced by N-Beats Algorithm on a Weekend

4.4.4 Discussion

The table 4.4 nested results displays the statistical results of the LSTM model on a weekday and a weekend using nested cross-validation. Overall, the weekday performed better in terms of both packet dropping accuracy and forecast node value errors. The average of the three models is calculated to obtain the comprehensive result of the LSTM model for both weekday and weekend node values forecasting. The weekday LSTM model achieved a respectable packet dropping prediction with 78.9% accuracy when compared to the weekend LSTM model's packet dropping predicting ability which is 67.3% accurate. The Weekday LSTM model performed well in all three of the splits almost uniformly. According to weekend LSTM model outcomes, the third split result impoverished the overall model's end result.

Table 4.4: Nested Cross-validation Results of LSTM Model

$Weekday$	$SMAPE$	MAE	$Accuracy$
$Split1$	0.251	4	80.4%
$Split2$	0.306	6	79.4%
$Split3$	0.348	9	76.8%
$Average$	0.302	6	78.9%
$Weekend$	$SMAPE$	MAE	$Accuracy$
$Split1$	0.311	5	77.3%
$Split2$	0.111	2	62.4%
$Split3$	1.011	28	62.2%
$Average$	0.477	12	67.3%

Individual results of each forecasting approach are displayed in the table 4.5. Even though the persistence model demonstrates the decent capabilities to forecast node values, they are the same as their preceding node values. Therefore, this model becomes unfit for forecasting if there are more variations in the node values. On the other hand, the LSTM model excellently captured the fluctuations in the node values when compared to the N-Beats model. Unexpectedly, the N-Beats model failed to grasp the variations in the node values as well as the node values themselves. The aggregated result of the LSTM model on a weekday is by far the best model among the conducted experiments. Both the LSTM models of weekday and weekend overcame the accuracy of packet dropping prediction of all other forecasting models used in this research.

Table 4.5: Comparison of Three Forecasting Approaches

$Weekday$	$SMAPE$	MAE	$Accuracy$
$PersistenceAlgorithm$	0.040	1	58.7%
$LSTMModel$	0.302	6	78.9%
$N-BeatsModel$	0.811	21	62.3%
$Weekend$	$SMAPE$	MAE	$Accuracy$
$PersistenceAlgorithm$	0.037	1	62.1%
$LSTMModel$	0.477	12	67.3%
$N-BeatsModel$	0.730	21	56.5%

.

4.5 Scheduling & Energy Models Simulation and Performance Evaluation

4.5.1 CPU Utilization Pre-simulation

This section finally gives an insight into the observed results to confirm that CPU utilization of an idle CPU is more than the CPU utilization time to switch between on and off states of VMs. Since the energy consumption of a VM is directly proportional to the CPU utilization, The researcher simulates the current study using Amazon Web Services (AWS) virtual machines. This allows the researcher to observe the difference in CPU utilization when VM kept idle and when VM was powering on and off at a regular interval [41] [33] . All the experiments are observed in virtual machines configured with 8 CPUs, 32 GB RAM, with up to 5 GB rate of data transfer. Figure 4.20 shows the status of the virtual machine when they are idle. The line graph illustrates how CPU utilization changes with time. The researcher observed that there is almost a constant CPU utilization with no considerable variation in the graph, while the CPU utilization fluctuates between 0.5 and 1 percent with time.

The Figure 4.21 illustrates the CPU utilization over time when the virtual machine is changing status from active to suspend at fixed intervals. The connected dots are the CPU utilization for an individual task. Although initially, there is a steep increase of up to 40 percent for the VM to configure, the rest of the graph is widespread during the time ranging between 0 and 15 percent of CPU utilization. Yet, there is no substantial CPU utilization

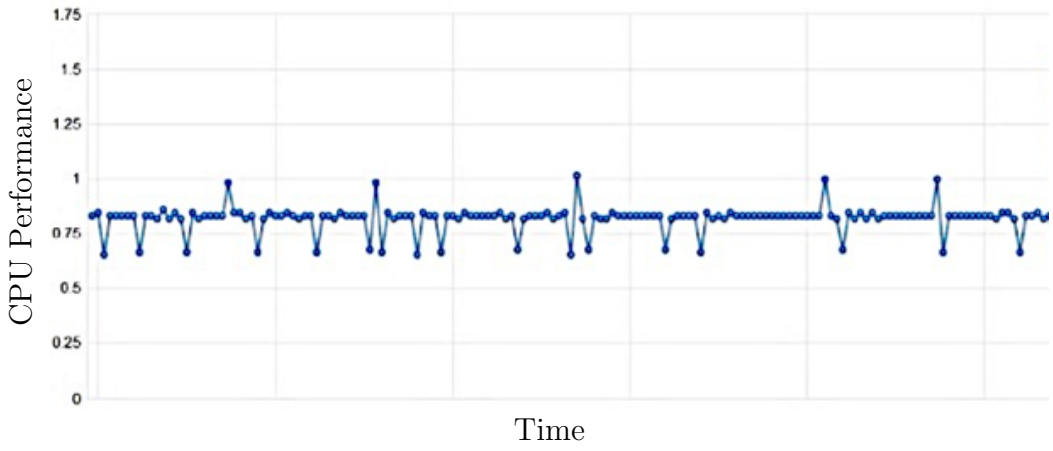


Figure 4.20: CPU Utilization with Idle VM

between any two consecutive suspended statuses. Figure 4.21 clearly shows that there is less CPU utilization as compared to those demonstrated in Fig. 4.20.

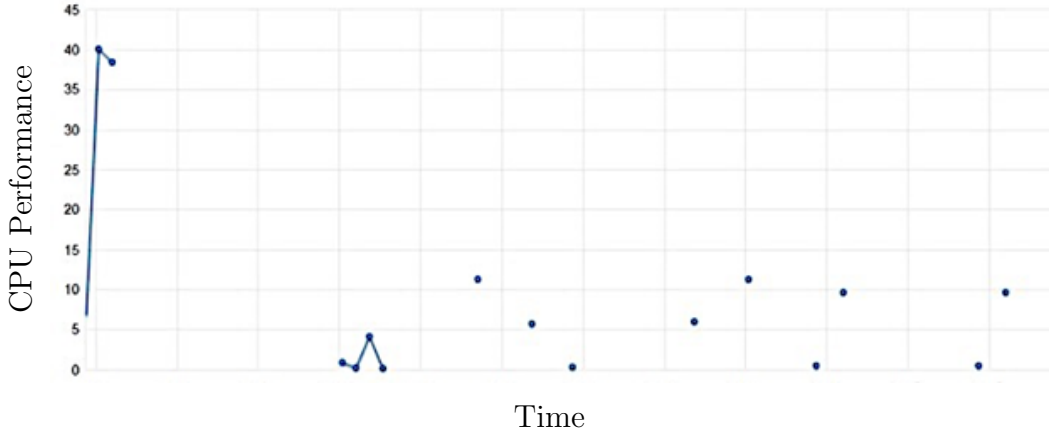


Figure 4.21: CPU Utilization with Powering On and Off a VM

4.5.2 Simulation Methodology

The scheduling strategy is demonstrated using the CloudSim simulator. CloudSim aims to simulate cloud environments and its features. It allowed us to fully customize the cloud completely within various levels. Massive community support for CloudSim is also a core

reason to choose.

4.5.3 Simulation Configurations

Our simulation is conducted within the CloudSim 4.0 environment in a windows 10 operating system. The system is running on a laptop with Intel(R) Core (TM) i7-8750H CPU @ 2.20GHz and 16-GB RAM. CPU utilization experiments are conducted using Amazon EC2 services with a basic instance of 1 vCPU and 1GB RAM.

4.5.4 Simulation Results

Figure 4.22 illustrates how the time to execute changes with the size of the task when the CPU frequency is constant. The researcher dynamically change the input task size parameter to observe the time to execute for a given task. The time to execute factor is shown in the x-axis, and the size of the task is expressed as the y-axis. The smaller the size of the task is, it hardly takes any time. As the size is increasing, the slope remains constant, because the task size is very small; therefore, a VM is underwhelmed and completes the task in a very small time. This plateau continues until a sudden spike happens, where the task is utilizing the most available resources. There is another sharp change when the VM is overwhelmed by the size of the task and the task has to wait until the last of its sub-task instructions are executed.

The researcher examines the impact of size on the *VM* with the help of Figure 4.23, as to how a *VM* handles a batch of tasks while maintaining the constant execution time. This line graph maintains a constant slope showing that as the size of the task increases, it is more likely the number of instructions required to execute per task increases at a constant rate. MIPS is the metric used to measure the *VM's* speed because MIPS is directly proportional to CPU cycles. Figure 4.23 proves that batch scheduling is an optimum solution for the best performance.

The results demonstrated in Figures 4.22 and 4.23 show that when applying a hybrid algorithm performs much better, as expected. It should be noted that our proposed strategy is functioning effectively because it is working better with small data sizes, considering the traditional acknowledgement packet size.

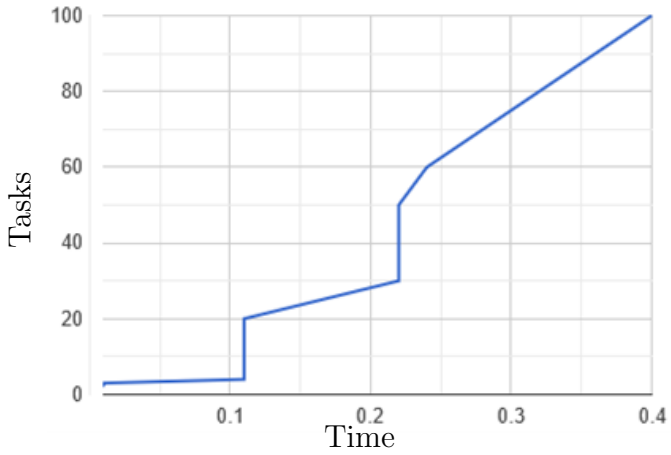


Figure 4.22: Time to Execute the Tasks Using VTA

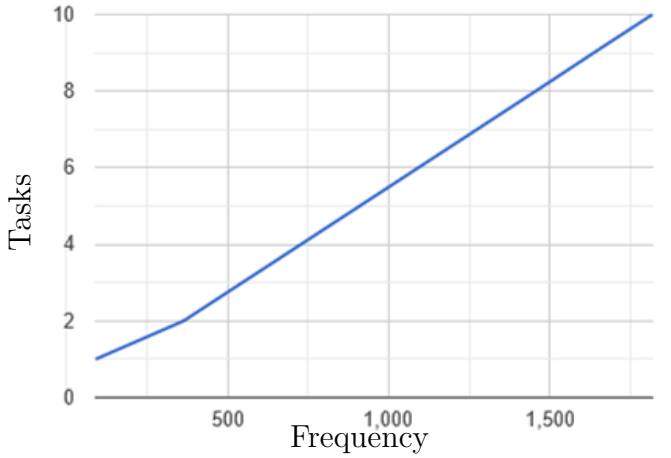


Figure 4.23: Frequency of VMs Using VFA

Chapter 5

Conclusion and Future Work

This chapter summarises the work done in this thesis and presents the theory and results of the experiments reported in Chapters 3 and 4. We also discuss potential possibilities for continued research on machine learning models for packet dropping prediction tasks using cloud and acknowledgment packets.

5.1 Conclusion

This thesis has introduced a novel architecture called Cloud Acknowledge Scheme that must deal with the packet droppings by the intermediate routers. The researcher designed several algorithms called Sender-site Algorithm, Cloud-site Algorithm, Intermediate Node-site Algorithm, Variable Time Algorithm and Variable Frequency Algorithm to monitor nodes and their transmissions in a network, as well as record the outcomes of the transmissions. Additionally, the acknowledgement packet structure has been designed and modified to incorporate some additional information that helps in identifying the location of the data packet during packet transmission. Moreover, an energy-aware hybrid scheduling strategy for the cloud is designed and tested by the investigator to accommodate a massive number of transmissions across the network.

Empirically, the current work emphasizes proving the existence of possibilities to predict the transmission outcomes in a network. One solution that has been explored in this research is to utilize machine learning techniques to predict a transmission outcome. Thus, it has

been determined that with enough history of transmission outcomes of an intermediate node, we can predict the outcome of a transmission that passes through a particular intermediate node. In order to validate the proposed theory, there is a need to collect a node's transmission history, Subsequently, train and evaluate machine learning models. In conclusion, the results accomplished by the developed machine learning models support the hypothesis of this research.

5.2 Future Work

The researcher also does have a future plan, an extention to the current work, to implement CACKS architecture in a network simulator that can utilize high-level machine learning models. Current CACKS design only considers a situation where an intermediate node does not respond. In the extention to this research, CACKS can be redesigned to consider multiple scenarios of packet dropping. Similarly, scheduling strategy can include various green computing techniques to reduce the energy consumption. Eventually, it also has been planned to evaluate this approach in several real-world nodes and in several large networks. However, the researcher's current best model predicts the transmission outcome with around 78.9% accuracy, most of this accuracy is due to the fact that the model is trained using the data that has a high drop rate. Therefore, the researcher expects to test the present approach, implemented in the current study, with a variety of nodes that has different drop rates. Present machine learning models use univariate data to forecast the outcome of future transmissions. However, the dataset lacks several key features like time, date, packet size, transmission protocol, etc. Therefore, it is preferred to explore multivariate datasets to train and evaluate this research's machine learning models.

Bibliography

[1] An Introduction to Machine Learning, . URL https://monkeylearn.com/machine-learning/.

[2] Introduction to Machine Learning and Different types of Machine Learning Algorithms | Hacker Noon, . URL https://hackernoon.com/introduction-to-machine-learning-and-different-types-of-machine-learning-algorithms-3z57i242u.

[3] Virtualization in Cloud Computing - javatpoint, . URL https://www.javatpoint.com/virtualization-in-cloud-computing.

[4] What is Jitter? | Nextiva Support, . URL https://www.nextiva.com/support/articles/what-is-jitter.html.

[5] What is a packet? | Network packet definition, . URL https://www.cloudflare.com/learning/network-layer/what-is-a-packet/.

[6] Open Shortest Path First (OSPF) Protocol fundamentals, April 2018. URL https://www.geeksforgeeks.org/open-shortest-path-first-ospf-protocol-fundamentals/.

[7] Virtualization in Cloud Computing - Benefits & Types of Virtualization, December 2018. URL https://data-flair.training/blogs/virtualization-in-cloud-computing/.

[8] What is MTR and Why is it Useful? A Guide to the MTR Network Monitor, April 2019. URL https://www.comparitech.com/net-admin/what-is-mtr/.

[9] What is an IP Address?, August 2019. URL https://whatismyipaddress.com/ip-address.

[10] Minimum spanning tree, May 2021. URL https://en.wikipedia.org/w/index.php?title=Minimum_spanning_tree&oldid=1025043917.

[11] Vaishali Advani. What is Machine Learning | Definition, Tools, how it Works & Uses, March 2021. URL https://www.mygreatlearning.com/blog/what-is-machine-learning/.

[12] Ankit. TCP flags, August 2019. URL https://www.geeksforgeeks.org/tcp-flags/.

[13] Raouf Boutaba, Mohammad A. Salahuddin, Noura Limam, Sara Ayoubi, Nashid Shahriar, Felipe Estrada-Solano, and Oscar M. Caicedo. A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, 9(1):16, June 2018. ISSN 1869-0238. doi: 10.1186/s13174-018-0087-2. URL https://doi.org/10.1186/s13174-018-0087-2.

[14] Jason Brownlee. How to Normalize and Standardize Time Series Data in Python, December 2016. URL https://machinelearningmastery.com/normalize-standardize-time-series-data-python/.

[15] Jason Brownlee. What Is Time Series Forecasting?, December 2016. URL https://machinelearningmastery.com/time-series-forecasting/.

[16] Jason Brownlee. Difference Between Classification and Regression in Machine Learning, December 2017. URL https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/.

[17] Jason Brownlee. How to use Data Scaling Improve Deep Learning Model Stability and Performance, February 2019. URL https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/.

[18] R. W. Callon. *RFC1195: Use of OSI IS-IS for routing in TCP/IP and dual environments*. RFC Editor, 1990.

[19] Sapna Chaudhary and Rahul Johari. Oruml: Optimized routing in wireless networks using machine learning. *International Journal of Communication Systems*, 33(11):e4394, 2020. doi: https://doi.org/10.1002/dac.4394. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4394`. e4394 dac.4394.

[20] Cisco. 2018 - 2023 cisco annual internet report white paper, 2020.

[21] Cloudflare. What is routin? — ip routing, 2021. URL `https://www.cloudflare.com/learning/network-layer/what-is-routing/`.

[22] Ascend Communications. OSPF Routing, 1999. URL `https://www.ip-sa.pl/doc/dslam/maxtnt/netcfg/tntospf.htm`.

[23] Wikipedia contributors. Residual neural network, 2020. URL `https://en.wikipedia.org/w/index.php?title=Residual_neural_network&oldid=978101817`. Visited on 15th March 2021.

[24] Shailey Dash. An Overview of Time Series Forecasting Models Part 1: Classical Time Series Forecasting Models, March 2020. URL `https://shaileydash.medium.com/an-overview-of-time-series-forecasting-models-part-1-classical-time-series-forecasting-models-2d877de76e0f`.

[25] Eswar M. Eduri. Forwarding information base lookup method, 2009. URL `https://patents.google.com/patent/US7606236B2/en`.

[26] Noami Elegant. The Internet Cloud Has a Dirty Secret, 2019. URL `https://fortune.com/2019/09/18/internet-cloud-server-data-center-energy-consumption-renewable-coal/`.

[27] Richard Fox. TCP big window and NAK options. RFC 1106, June 1989. URL `https://rfc-editor.org/rfc/rfc1106.txt`.

[28] Mohamed A. El Galil. Network Routing Optimization Using Swarm Intelligence. *arXiv:1209.3909 [cs]*, August 2015. URL `http://arxiv.org/abs/1209.3909`.

[29] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya. Environment-conscious scheduling of HPC applications on distributed Cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71:732–749, 2011. ISSN 0743-7315. doi: https://doi.org/10.1016/j.jpdc.2010.04.004.

[30] Tim Greene. Architecture of the Internet, March 2020. URL `https://www.tutorialspoint.com/Architecture-of-the-Internet`.

[31] Edward T. Grochowski and Murali Annavaram. Energy per instruction trends in intel ® microprocessors. 2006.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL `http://arxiv.org/abs/1512.03385`. arXiv: 1512.03385.

[33] Chris Hoffman. Psa dont shut down your computer just use sleep or hibernation, 2017. URL `https://www.howtogeek.com/256395/psa-don%E2%80%99t-shut-down-your-computer-just-use-sleep-or-hibernation/`. Visited on 25th October 2019.

[34] Tzveta Iordanova. An introduction to stationary and non-stationary processes. URL `https://www.investopedia.com/articles/trading/07/stationary.asp`.

[35] Monalisa Jena, Ranjan Kumar Behera, and Santanu Kumar Rath. Machine Learning Models for Stock Prediction Using Real-Time Streaming Data. In Satchidananda Dehuri, Bhabani Shankar Prasad Mishra, Pradeep Kumar Mallick, Sung-Bae Cho, and Margarita N. Favorskaya, editors, *Biologically Inspired Techniques in Many-Criteria Decision Making*, Learning and Analytics in Intelligent Systems, pages 101–108. Springer International Publishing, 2020. ISBN 978-3-030-39033-4. doi: 10.1007/978-3-030-39033-4_10.

[36] Nicola Jones. Residual neural network, 2018. URL `https://www.nature.com/articles/d41586-018-06610-y`. Visited on 1st December 2019.

[37] Nan Kang. *Detecting Misbehaving Nodes in Mobile Ad hoc Networks*. PhD thesis, Acadia University, Wolfville, NS, February 2011.

[38] kaspersky. What is an IP Address Definition and Explanation, January 2021. URL https://www.kaspersky.com/resource-center/definitions/what-is-an-ip-address.

[39] Kartik Khare. Deploying ML Models in Distributed Real-time Data Streaming Applications, January 2020. URL https://towardsdatascience.com/deploying-ml-models-in-distributed-real-time-data-streaming-applications-217954a0b423.

[40] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biol. Cybern.*, 43(1):59–69, January 1982. ISSN 1432-0770. doi: 10.1007/BF00337288. URL https://doi.org/10.1007/BF00337288.

[41] Ying Lu Leping Wang. Efficient power management of heterogeneous soft realtimeclusters. In *IEEE Real Time Syst Symp*, 2008. doi: https://doi.org/10.1109/RTSS.2008.31.

[42] Eryk Lewinson. Choosing the correct error metric: MAPE vs. sMAPE, November 2020. URL https://towardsdatascience.com/choosing-the-correct-error-metric-mape-vs-smape-5328dec53fac.

[43] Ke Liang and Mitchel Myers. Machine Learning Applications in the Routing in Computer Networks. *arXiv:2104.01946 [cs]*, April 2021. URL http://arxiv.org/abs/2104.01946. arXiv: 2104.01946.

[44] Anudeep Mangu. Managing Energy Consumption of Data Centers. URL http://large.stanford.edu/courses/2018/ph240/mangu2/.

[45] Majkowski Marek. When TCP sockets refuse to die, September 2019. URL https://blog.cloudflare.com/when-tcp-sockets-refuse-to-die/.

[46] Mullins Michael. Exploring the anatomy of a data packet, July 2001. URL https://www.techrepublic.com/article/exploring-the-anatomy-of-a-data-packet/.

[47] Microsoft. What is IaaS Infrastructure as a Service | Microsoft Azure, . URL https://azure.microsoft.com/en-ca/overview/what-is-iaas/.

[48] Microsoft. What is PaaS Platform as a Service | Microsoft Azure, . URL https://azure.microsoft.com/en-ca/overview/what-is-paas/.

[49] Microsoft. What is SaaS Software as a Service | Microsoft Azure, . URL https://azure.microsoft.com/en-ca/overview/what-is-saas/.

[50] Bogdan Militaru. How to build low-cost, real-time and scalable machine learning models in 4 steps using Google Cloud, April 2021. URL https://towardsdatascience.com/how-to-build-low-cost-real-time-and-scalable-machine-learning-models-in-4-steps-using-google-cloud-ea2cff85cca2.

[51] J Moy. rfc2328, April 1998. URL https://datatracker.ietf.org/doc/html/rfc2328.

[52] Christopher Olah. Understanding lstm networks, 2015. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Visited on 27th October 2020.

[53] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados, and Yoshua Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1ecqn4YwB.

[54] Blazej Osinski and Konrad Budek. What is reinforcement learning The complete guide, July 2018. URL https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/.

[55] Marius Paulescu, Eugenia Paulescu, and Viorel Badescu. Chapter 9 - Nowcasting solar irradiance for effective solar power plants operation and smart grid management. In Ravinesh Deo, Pijush Samui, and Sanjiban Sekhar Roy, editors, *Predictive Modelling for Energy Management and Power Systems Engineering*, pages 249–270. Elsevier, 2021. ISBN 978-0-12-817772-3. doi: 10.1016/B978-0-12-817772-3.00009-4. URL https://www.sciencedirect.com/science/article/pii/B9780128177723000094.

[56] Jun Pei, Xinbao Liu, Wenjuan Fan, Panos M. Pardalos, and Shaojun Lu. A hybrid BA-VNS algorithm for coordinated serial-batching scheduling with deteriorating jobs,

financial budget, and resource constraint in multiple manufacturers. *Omega*, 82:55–69, 2019.

[57] Larry L Peterson and Bruce S Davie. *Computer Networks: A Systems Approach*. 2012.

[58] Radhika. Choosing Evaluation Metrics For Classification Model, October 2020. URL `https://www.analyticsvidhya.com/blog/2020/10/how-to-choose-evaluation-metrics-for-classification-model/`.

[59] Brian Ray. Link labs: What is m2m, 2018.

[60] Jennifer Rexford. Route Optimization in IP Networks. In Mauricio G. C. Resende and Panos M. Pardalos, editors, *Handbook of Optimization in Telecommunications*, pages 679–700. Springer US, 2006. ISBN 978-0-387-30662-9 978-0-387-30165-5. doi: 10.1007/978-0-387-30165-5_24. URL `http://link.springer.com/10.1007/978-0-387-30165-5_24`.

[61] Ting Shi, Mei Yang, Xiang Li, Qing Lei, and Yingtao Jiang. An energy-efficient scheduling scheme for time-constrained tasks in local mobile clouds. *Pervasive and Mobile Computing*, 27:90–105, 2016. ISSN 1574-1192. doi: https://doi.org/10.1016/j.pmcj.2015.07.005.

[62] Soumya Shrivastava. Cross Validation in Time Series, January 2020. URL `https://medium.com/@soumyachess1496/cross-validation-in-time-series-566ae4981ce4`.

[63] Senthilkumar Subramaniyan, William Johnson, and Karthikeyan Subramaniyan. A distributed framework for detecting selfish nodes in MANET using Record- and Trust-Based Detection (RTBD) technique. *J Wireless Com Network*, 2014(1):205, December 2014. ISSN 1687-1499. doi: 10.1186/1687-1499-2014-205.