Investigation of Fast Uplink Grant for Machine Type Communication

Undergraduate Project Report

**** Draft Version ****

By

Sankalpa V.L.T. EG/2015/2757

Abstract

Machine type communication (MTC) will lead to remarkable and rapid developments in a wide range of domains while evolving the nature of different environments and markets. It will connect a large number of MTC devices to the internet and other networks, forming the ecosystem called the Internet of Things. International telecommunication union (ITU) has defined, enabling massive MTC over cellular networks as one of the key features of forthcoming 5G based international mobile telecommunications (IMT) -2020 networks. MTC networks must be able to support ultra-reliable, low latency communication and intelligently manage machine type devices (MTD) in real-time dynamic environments. According to the existing literature, the concept of fast uplink grant shows potential to realize above requirements and it is emerging as a promising solution for enabling massive machine type communication (MTC) in the Internet of Things over cellular networks. The key intelligent features of base station (BS) that are required for adapting the fast uplink grant, which are prediction of Machine Type Devices which have data packets to transmit and the optimal allocation of the fast uplink grant can be achieved by integrating notions of artificial intelligence (AI) and machine learning. This project focuses on proposing an innovative learning framework for adapting to fast uplink grant in cellular-enabled massive MTC using machine learning and deep learning concepts.

Preface

This report has been prepared in order to provide details and description of the undergraduate project, Investigation of Fast Uplink Grant for Machine Type Communication, which was conduct as per the partial fulfillment of the requirements for the Degree of Bachelors of Science in Engineering. The first chapter of the report includes information about machine type communication and fast uplink grant concepts along with problem formulation, objectives, scope, research methodology, and time plan of the project. The second chapter gives information about the previous work and theories relevant the project while highlighting the requirement of this project by describing the weakness and limitation of existing work. The third chapter expresses the proposed fast uplink grant framework along with the detailed descriptions on implemented machine learning models and performance evaluations. The fourth chapter gives information about results and analysis carried out with proposed fast uplink grant framework. Finally, summary and the conclusion along with recommendations and future works required to realized fast uplink grant in real word.

Acknowledgement

This undergraduate project was a great opportunity for my professional growth, development and specialization toward data science and wireless communication sectors. I am grateful for having a chance to expose to industry trending technologies like could computing and Python data analysis while working on the project. Therefore, I wish to express my gratitude to all those people who helped me towards the success of my undergraduate project.

First, I would like to express my deepest appreciation to my project supervisor Dr. C. K. W. Seneviratne, external-supervisors Prof. Nandana Rajatheva and Dr. Samad Ali, Center for Wireless Communications (CWC), University of Oulu, Finland. Also, I would like to extend my gratitude to the members of undergraduate project evaluation panel, Department of Electrical and Information Engineering, University of Ruhuna for supervise our undergraduate project activities and for their guidance in making this project a success.

Furthermore, I would also like to acknowledge Dr. Rasika Withanawasam, Chief Architect, Surveillance Systems at LSEG Technology who encourages and motivates us to use deep learning techniques for the project. Finally, we would like to thank all other lecturers of Department of Electrical and Information Engineering for their kind support for this project.

Contents

List of Figures

1. INTRODUCTION

Machine-type communications (MTC) empowers a broad range of applications from mission-critical services to the massive deployment of autonomous systems. To expand these applications broadly, cellular networks are recognized as a possible method to provide connectivity for MTC devices. The universal deployment of these systems decreases network installation costs and provides mobility support. Therefore, MTC has excellent potential in a wide range of applications and services. As illustrated in figure 1.1 the potential applications are widespread across different industries, including healthcare, logistics, manufacturing, process automation, energy, and utilities.

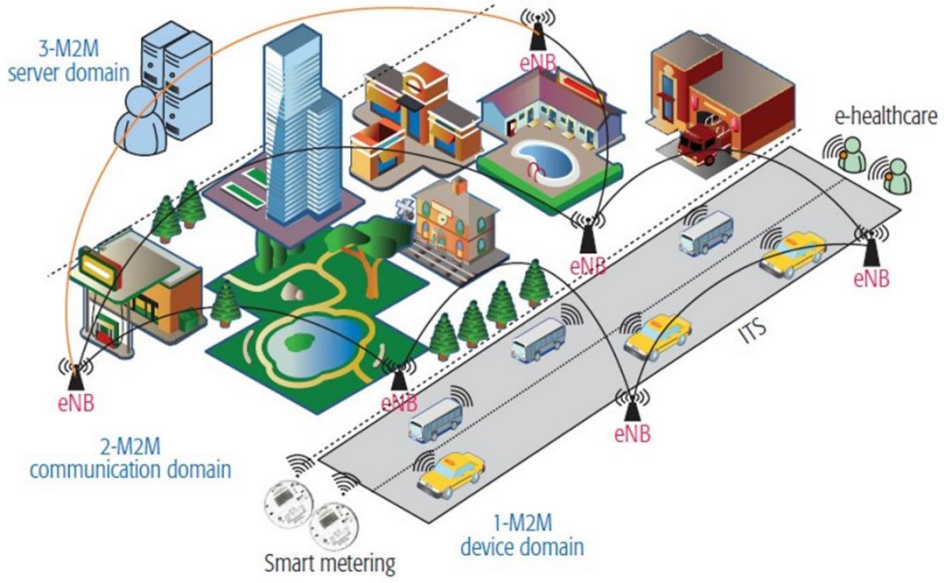


Figure 1.1 Massive Machine Type Communication in 5G and beyond network

1.1 Background

However, based on the service requirements of MTC, there are key challenges that currently limit the broad use of cellular networks for MTC. Continuously cellular networks have been mainly designed and optimized to serve traffic from human-to-human (H2H) communications, which are generally characterized by bursts of data during active periods with a higher demand for downlink. However, major MTC applications have different traffic characteristics: normally small and infrequent data generated from a mass of MTC devices imposing a higher traffic volume on the uplink. As tabulated in table1.1 examples of these infrequent data transmissions, which are uplink centric, include advanced metering infrastructure and vending machines.

Table 1.1 Examples of MTC applications and their requirements.

Service functions	Application examples	Main requirements
Metering	Electric power, gas, and water metering	Support of a massive number of MTC devices with small data bursts and high coverage
Control systems and monitoring	Industrial and home automation, and real-time control	High mobility and low-latency data transmissions
Payment	Point of sale and vending machines	High level of security

Security and public safety	Surveillance systems, home security, and access control	High reliability, high security, and low latency

Regarding existing cellular networks the initial step required to establish an uplink connection between User Equipment (UE) and a base station (BS) is random access (RA). According to the first step of the RA process BS broadcast information about available random access channels (RACH). When UE requires an uplink connection to transmit data, each UE randomly selects one of the available RACH to send a scheduling request. RA process exhibits an inability to support massive cellular-enabled MTC due to the following problems presented in table 1.2.

Table 1.2 Problems associated with RA process in massive MTC

Problems of RA process	Description
Collisions	➢ Number of MTDs competing for RA slots is larger than the number of RA opportunities ➢ More than one MTD select same RA slot and collisions will occur ➢ Probability of RA scheduling request collisions will increase
Delays	➢ BS unable to decode the collided requests ➢ MTDs have to re-send the request and ultimately this will case for longer delays ➢ RA signaling itself takes time to establish uplink transmission
Signaling overhead	➢ Signaling overhead of the RA process is larger than the actual packet size that intended to transmit ➢ RA process is highly ineffective for cellular-enabled massive MTC

Therefore, to develop innovative wireless medium access schemes for cellular-enabled MTC fast uplink grant has proposed by 3rd Generation Partnership Project (3GPP) release 11. Fast uplink grant allows BS to select MTDs, which has data to transmit and actively allocate them uplink resources using one broadcast signal for the entire cell. Hence, whenever MTD has data to transmit, it can wait predefined time until BS allocates fast uplink grant. Since fast uplink grant does not send any scheduling requests, it permits to use RA resources for uplink direction transmission, satisfy QoS requirements of MTDs by actively allocating fast uplink grant for MTDs with stricter latency requirements.

In order to adapt to fast uplink grant, the BS must implement superior traffic prediction mechanisms to predict the set of MTDs that have data to transmit. Most of the existing systems on wireless network traffic modeling for MTC is concentrated on estimating the number of devices or the number of packets entering in the system. However, source traffic modeling is a different dilemma since we are interested in exactly predicting which MTDs will enter the network [4]. In essence, at each time slot, the BS needs to predict which MTDs will have traffic to send. Even the majority of MTDs in the idle mode most of the time, source traffic prediction becomes more difficult. However, such predictions for idle mode MTDs are generally possible in MTC networks due to the following reasons.

- Most of the MTDs are stationary or exhibit low mobility. Therefore the set of MTDs communicating with a BS is often fixed.
- Furthermore, prediction can be done using a centralized controller that keeps track of the BSs and their associated MTDs.

Once predictions are accurately performed, the BS further needs to be able to determine the allocation order of MTDs. This challenge is especially raised when the number of devices significantly exceeds the number of resources. Therefore, advanced allocation algorithms are needed to enable an optimal selection of MTD for fast uplink grant.

1.2 Problem Statement

From the literature-survey of the project, it was possible to recognize a set of shortcomings and limitations of existing frameworks for adapting to a fast uplink grant. One of the main challenges there is, accurately predicting the MTDs which have data to transmit at any given time as inaccurate predictions will lead to resource wastage. The problem with the currently implemented traffic prediction framework is its accuracy has not been verified. Additionally, it cannot generate a solid prediction due to its nature of the output, scalability issues and complexity. Therefore, there is room for further development of the currently suggested method for traffic prediction. More critical shortcomings of this method have been described in chapter 2 and chapter 4.

Another main challenge of the Fast Uplink Grant is the optimal allocation of uplink channels to the MTDs which have data packets to transmit considering their latency requirements. The stochastic nature of the latency requirements of MTD can exacerbate this problem. This can potentially lead to unreliable and high latency MTC transmissions. There are few channel allocation methods suggested but only one of those methods has been implemented and analysed. Hence, there is room for enhancement of the uplink channel allocation by investigating other methods. Furthermore, the currently proposed optimal channel allocations framework does not use the output of the source traffic prediction framework. Instead of actual traffic prediction framework output, it has considered a separately generated artificial MTD availability dataset for channel allocations. Therefore, there is void in studies of how the traffic prediction and optimal allocations can be implemented together.

1.3 Objectives

The overall objective of this project is to introduce an innovative machine learning framework that can contribute to the realization of Fast Uplink Grant by overcoming the weaknesses of existing methods. The main research objectives of this project fall under the three categories of sub-tasks.

- Evaluating the performance of the existing traffic prediction framework by verifying its accuracy, limitations, shortcomings, and any possibility of further improvements.
- Implement a novel traffic forecasting mechanism that capable of precisely predicting which MTDs will enter the network at each time slot using machine learning and deep learning techniques.
- Perform modification on existing optimal channel allocation framework that required facilitating optimal channel allocation based on the output of traffic prediction.

1.4 Research Methodology

This section describes the methodology this undergraduate project in terms of research methods and logical steps as follows.

1. **Literature survey**

As the initial step of this research work, a literature survey conducted to obtain the essential understanding of machine type communication, existing source traffic prediction and optimal grant allocation frameworks. Technical papers, IEEE conference papers, journal articles, 3GPP/ITU documents, and books published related to the research areas were investigated throughout this step of the project.

2. **Implementation of existing Directed Information (DI) based traffic prediction framework**

During the second step of this project, a MATLAB program was created to replicate the existing DI algorithm based on the DI conference paper [] and the literature. After that, MATLAB program and implementation details were sent to the authors of the original DI paper to verify the validation of the implementation. By finishing this step, it was able to achieve the necessary technical skill required to develop or further analysis of this approach.

3. Performance evaluation and analysis on implemented DI based traffic prediction framework

Evaluating the performance of the earlier implemented DI framework by verifying its accuracy. More under this step, the analysis was continued to draw a clear line between limitations and any feasibility of further improvements on the directed information algorithm.

4. Survey on feasible machine learning techniques for source traffic prediction

This survey was conducted as a primary step of developing a novel source traffic forecasting mechanism to overcome the critical shortcomings and scalability issues of the DI algorithm. This study focuses on the use of machine learning and deep learning methods such as Hidden Markov Model (HMM), Artificial Neural Networks (ANN), Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM) for time series forecasting.

5. Build simulation program for transmission data generation

Simple Python program was written to generate MTD transmission data by modeling the transmission records of each MTD as a binary time sequence where packet transmission is presented by 1 and being silent with 0. This program uses to generate periodic, event-driven and random MTD transmission patterns. NumPy library uses to write this program because it supports multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

6. Propose and build novel source traffic forecasting framework

Under this step of the project, LSTM neural network-based source-traffic prediction mechanism was implemented. Furthermore, pre-processing of artificially generated transmission data, LSTM model training, validation, and hyperparameter optimization jobs also conducted under this step. Finally, model performance evaluation and testing are done in terms of resource wastage, RA request reduction, and prediction accuracy. TensorFlow backend Keras library and Amazon Sagemaker cloud machine-learning platform use to build this LSTM model.

7. Implementation of existing optimal channel allocation framework

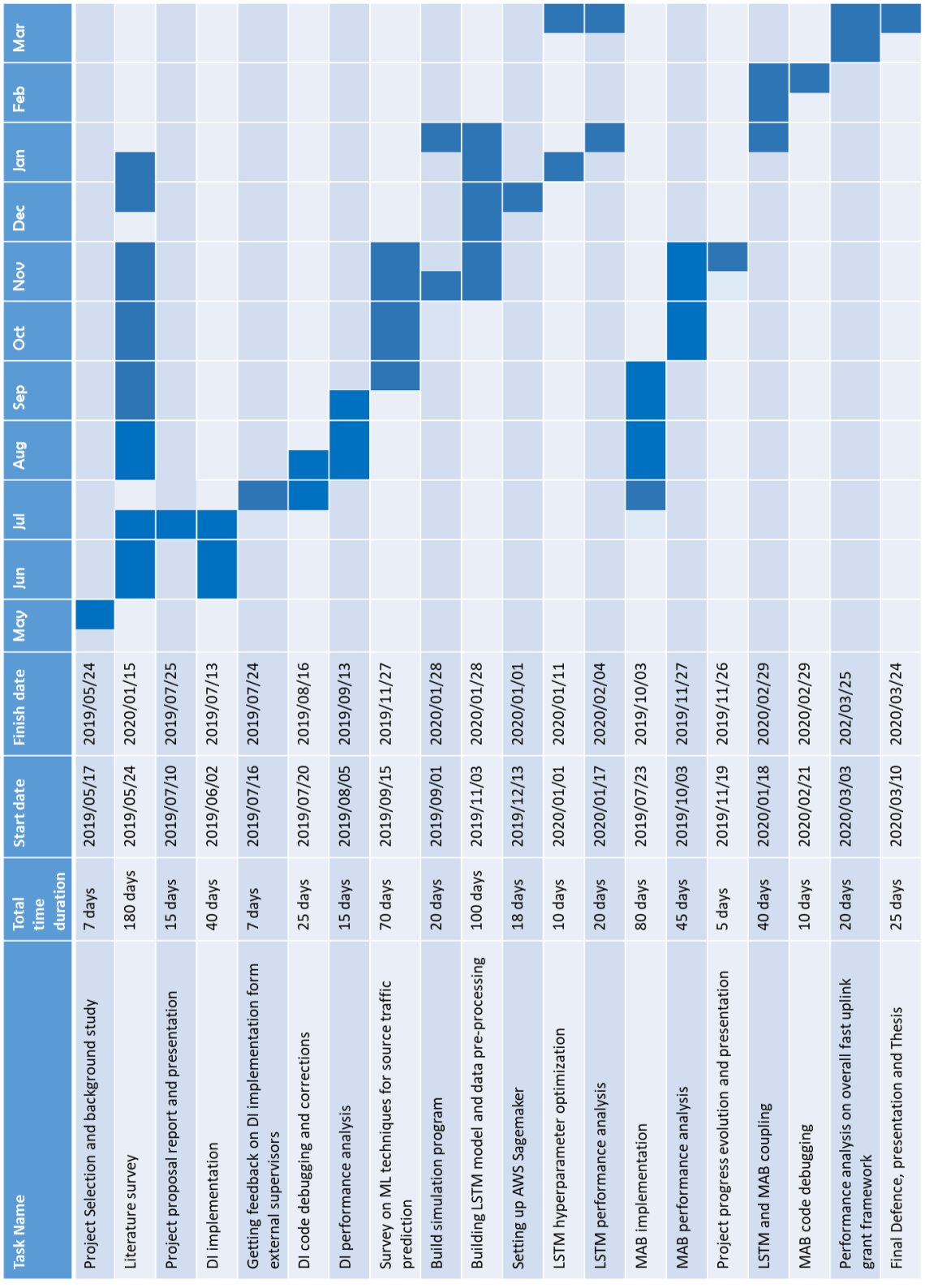
In this step of the project, Python program was created to replicate the current optimal channel allocation framework based on the sleeping multi-armed bandit conference paper []. Due to the unavailability of straightforward Python libraries to implement Multi-Armed Bandit (MAB) and Upper Confidence Bound (UCB) algorithms, both of them were implement using data pre-processing libraries (Pandas and Numpy) along with looping and decision making statements.

8. Modify the optimal allocation framework and couple with proposed traffic prediction framework

Under this step of the project, MTD data packet queuing mechanism was embedded to the earlier implemented optimal allocation framework. This permits selecting the optimal set of MTDs for channel allocation based on the output from the traffic prediction framework. The output of the traffic prediction framework takes the form of probabilities of MTDs has data packets to transmit at any given time step. After that, performance evaluation of the overall system was perform in terms of queuing delay using data manipulation methods available in Pandas and NumPy Python libraries. Further

analysis was performed to investigate the impact of traffic prediction accuracy on allocations and graphically interpret the results of performance evaluations using graph-plotting methods available in the Matplotlib library.

1.5 Work Plan and Time Line



Task Name	Total time duration	Start date	Finish date
Project Selection and background study	7 days	2019/05/17	2019/05/24
Literature survey	180 days	2019/05/24	2020/01/15
Project proposal report and presentation	15 days	2019/07/10	2019/07/25
DI implementation	40 days	2019/06/02	2019/07/13
Getting feedback on DI implementation form external supervisors	7 days	2019/07/16	2019/07/24
DI code debugging and corrections	25 days	2019/07/20	2019/08/16
DI performance analysis	15 days	2019/08/05	2019/09/13
Survey on ML techniques for source traffic prediction	70 days	2019/09/15	2019/11/27
Build simulation program	20 days	2019/09/01	2020/01/28
Building LSTM model and data pre-processing	100 days	2019/11/03	2020/01/28
Setting up AWS Sagemaker	18 days	2019/12/13	2020/01/01
LSTM hyperparameter optimization	10 days	2020/01/01	2020/01/11
LSTM performance analysis	20 days	2020/01/17	2020/02/04
MAB implementation	80 days	2019/07/23	2019/10/03
MAB performance analysis	45 days	2019/10/03	2019/11/27
Project progress evolution and presentation	5 days	2019/11/19	2019/11/26
LSTM and MAB coupling	40 days	2020/01/18	2020/02/29
MAB code debugging	10 days	2020/02/21	2020/02/29
Performance analysis on overall fast uplink grant framework	20 days	2020/03/03	202/03/25
Final Defence, presentation and Thesis	25 days	2020/03/10	2020/03/24

Figure 1.2 Time line

2. LITERATURE REVIEW

2.1 Machine Type Communication (MTC)

Machine Type Communications (MTC) or Machine to Machine (M2M) communication is about enabling communications among MTC devices and electronic devices or enabling communications from MTC devices to a central MTC server or a set of MTC servers. Communications can use both wireless and fixed networks. MTC will enable an endless number of applications in a wide plethora of domains, impacting different environments and markets. It will connect a potential number of MTC devices to the Internet and other networks, forming the 5G based Internet of Things (IoT) eco-system **[Machine Type Communications in 3GPP Networks].** As illustrated in **[figure 2.1],** billions of machines and industrial devices will be potentially able to benefit from MTC.

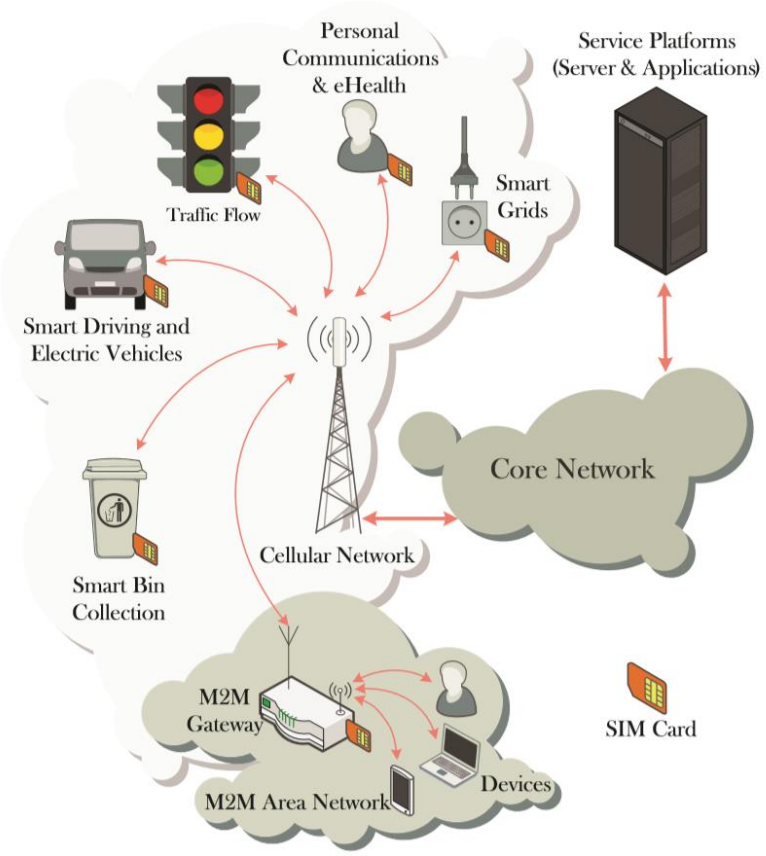


Figure 2.1 Typical MTC applications

With a wide range of potential applications, MTC is gaining tremendous interest and attention among mobile network operators, equipment vendors, MTC specialist companies, and research bodies. To facilitate convergence among these different stakeholders, 3rd Generation Partnership Project (3GPP) has started working on MTC ***[3GPP Service requirements for Machine-Type Communication].***

While some existing MTC deployments use short-range radio access technologies, deployment of MTC solutions based on mobile access technologies is vitally important to support a wide range of MTC devices, including those with mobility services.

Mobile access based MTC solutions is also more adequate for supporting MTC services that require immediate and reliable delivery of data to distant MTC servers. Exhibiting different behavior than ordinary mobile network terminals, MTC devices need to be treated differently to enable mobile

operators to accommodate a potential number of them. Hence, there is a need for the optimization of networking solutions, specifically tailored for MTC in mobile networks.

Regarding 3GPP, the first study on MTC was presented in ***[3GPP, "Facilitating Machine to Machine Communication in GSM and UMTS," TR 22.868, Mar. 2007],*** with no subsequent normative specifications. In Release 10, the 3GPP System Architecture working group 2 (SA2) is defining 3GPP network system improvements to support MTC in UMTS (Universal Mobile Telecommunications System) and LTE (Long Term Evolution) core networks. These network improvements and optimizations are mainly focused on overcoming the challenges of cellular-enabled massive MTC. Typical examples of these challenges are signaling congestions and network overloading.

2.1.1 MTC Architecture

Figure 2.2 illustrated a standard MTC network architecture. It consists of three main domains, namely the MTC device domain, the core network domain, and the MTC application domain. Current MTC core architecture consists of the most important nodes of a 3GPP Evolved Packet System (EPS) network are shown. Depending on the use case, the MTC devices transmit or receive a determined amount of data, as an example let's consider a smart meter sending measurement results every day at 10:00h. MTC devices can be either fix installed (as an example implemented in a factory's machine, gas meters, etc.) or mobile (as an example of fleet management devices in trucks). These different characteristics and service requirements of MTC devices (as examples in terms of mobility, amount of transmitted data, security, etc.) enables mobile operators to make different optimizations for grouping MTC devices, charging MTC applications, and controlling network resources used by the MTC devices. This core network domain can be a wired or wireless network. In this study, the major focus on the situation when the MTC core network is a 3GPP mobile network. Table 2.1 provides a brief description of the most important EPS nodes, shown in Figure 2.2, including the two newly defined entities, service capability servers (SCSs) and the MTC-interworking function (MTC-IWF). The MTC application domain consists of MTC servers, under the control of the mobile network operator or a third party.

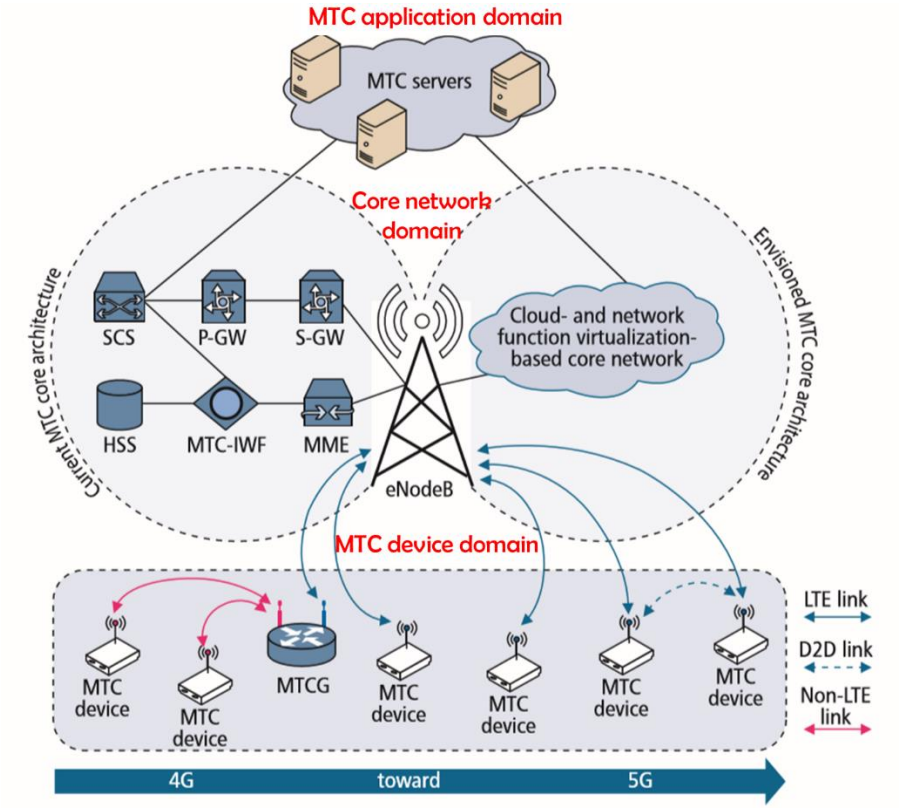


Figure 2.2 MTC network architecture

As in Figure 2.2, 3GPP has defined two kinds of communications for MTC applications: communications between an MTC device and a server, and communications between two MTC devices. The required connectivity for the MTC devices and the servers can be provided by cellular systems. The MTC devices can be connected to base stations (e.g. eNodeBs) directly or through MTC gateways (MTCGs). Example MTC applications for communications between an MTC device and a server are indoor health monitoring at home, water, gas, temperature or power metering. Other types of MTC applications associated with an alternative model of communication such as the peer-to-peer (P2P) model, where MTC devices are communicating directly among themselves.

Table 2.1 MTD core network nodes and functions

MTC core network nodes		Description
eNodeB	Evolved Node B	Act as base station for cellular-enabled MTC
MME	Mobility Management Entity	Act as control plane entity for all mobility-related functions, paging, authentication, bearer management in the EPS
HSS	Home Subscriber Server	The main database containing subscription-related information that important to provide connectivity
S-GW	Serving Gateway	Act as a local mobility anchor for intra-3GPP handoffs
P-GW	Packet Data Network Gateway	Interface MTC network with the packet data network like internet
SCS	Service Capability Servers	Mainly designed to offer services for MTC applications hosted in external networks
MTCG	MTC gateways	Exchange data between the cellular and capillary networks
MTC-IWF	MTC-Interworking Function	Hides the internal Public Land Mobile Network (PLMN) topology and relays or translates signaling protocols to invoke specific functionalities in PLMN

2.1.2 MTC Service Control

According to the 3GPP Technical Specification 22.368 (release version 11.5.0), MTC devices hold a unique set of service requirements. These requirements needed to be fulfilled as special services when enabling MTC over cellular networks in order to optimize the mobile network to support MTC applications. These MTC services are subscribed and controlled by the MTC subscription in the HSS. One subscription can be shared among several MTC devices, includes the security credentials used to authenticate the devices. MTC devices sharing the same MTC subscription can use all subscribed MTC services belonging to the said subscription. The MTC services are activated at any time that the MTC device's subscription is attached to the network and whenever service is supported by the network. During the course of the subscription, MTC subscribers have the flexibility to activate unsubscribed MTC services or deactivate subscribed MTC services based on the operator policies. There are already a set of MTC service requirements considered in 3GPP ***[3GPP, "Service Requirements for Machine-Type Communications," TS 22.368 V10.1.0, June 2010].*** Some of these service requirements are listed in Table 2.2.

A use case, explaining how services could be subscribed for specific MTC devices, could be "pressure-sensing MTC devices" on a train bridge: each time a train runs over the bridge, the MTC pressure devices do their measurements and provide the result to the corresponding MTC application server for continuous safety condition monitoring purposes of the bridge. These devices are mounted on the bridge so they would subscribe to the "low mobility" service to prevent unnecessary tracking area updates (TAUs), used by the MTC device to notify the network about its current location within the

network. The devices can also subscribe to the “small data transmission” service, “PS only” service, and the “mobile originated only” service since the devices initiate the transmissions. In case the bridge is rarely traversed by trains, then the “infrequent transmission” service would also be applicable.

Table 2.2 MTC service requirements in 3GPP

MTC Service Requirements	Description
Low mobility	Rarely moving, slow-moving and only moving within a certain area.
Periodic data transmissions	MTC data packets delivery only during predefined time intervals
Event-driven data transmissions	MTC data packets delivery due to occurrence of a particular event
Time tolerant	Data transfer can be delayed within some limited interval without harmfully impacting to the proper functionality of the application
Packet Switched (PS) only	MTC device supports only PS services
Small data transmissions Only	A small number of data packets are exchanged
Group based MTC service requirements	Possible to identify a group or cluster of MTC devices that have the same set of services requirements and characteristics
Correlated data transmission	Possible to identify correlations among the data transmissions of MTC device
Different Quality of Services (QoS)	Possible to prioritize the data packets based on importance, significance, and severity
Secure connection	A secure connection between MTC devices and MTC servers required
Random data transmissions	High entropy data transmission due to unpredictable changes in environmental conditions
Location-specific trigger	Triggers MTC devices in a particular area
Infrequent transmission	Long-period between two data transmissions

Because MTC services may be incompatible with each other, the network operator can restrict the subscription when incompatibility with already subscribed services is foreseen. Similarly, the network operator may also forbid attachment to the network when a mandatory service is not or cannot be enabled. Figure 2.3 illustrates the MTC service cycle and describes in steps the first solution for the service requirement issue that is recognized in 3GPP:

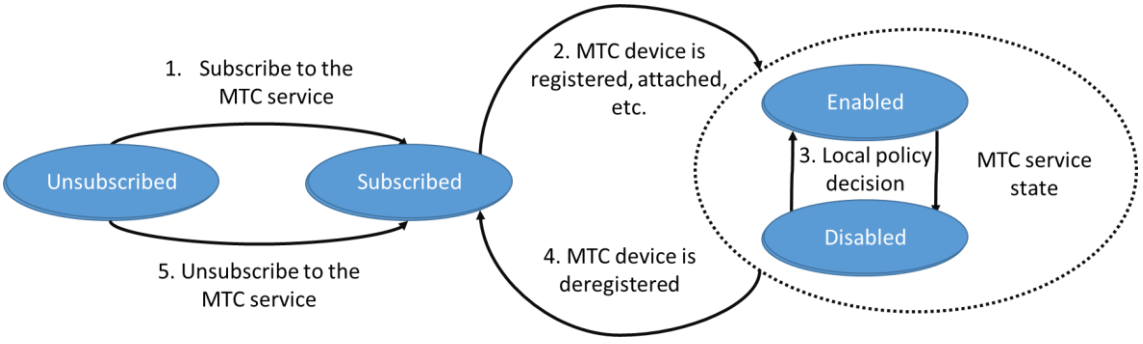


Figure 2.3 MTC service cycle

1) Based on a pre-configuration in the operator system (as an example HSS), MTC services are initially unsubscribed. A subscriber subscribes to an MTC service via an operator web interface. This MTC service is then marked as subscribed to the MTC subscription profile. The operator system (HSS) can always accept or reject the subscription request.

2) When an MTC device attaches and registers to the network, the subscribed services are downloaded from HSS to MME in the EPS or to the visitor location register (VLR) of SGSN in UMTS. The downloaded subscription services define the subscription profile. Unsubscribed services are simply not downloaded.

3) A MTC service can be in either "enabled" or "disabled" state. Initially, all subscribed services are considered as "enabled". However, based on a local policy, SGSN/MME can disable all or part of subscribed services depending on the serving network capability and/or MTC device capability.

4) When a MTC device deregisters from the network, the MTC services are still subscribed. This is also regardless of whether all subscribed services are disabled or not in SGSN/MME.

5) All subscribed services can be unsubscribed utilizing an operator interface regardless of whether they are enabled or disabled in the MME/SGSN.

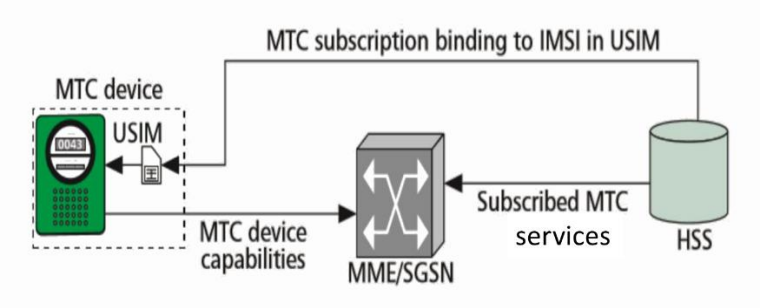


Figure 2.4 Main EPS network entities for MTC service control [ref]

Figure 2.4 shows the three main entities in the network that are involved in the MTC service control. The MTC subscription identifies subscribed MTC services in the HSS. The binding between subscription and device is done with the IMSI (international mobile subscriber identity) of the USIM (universal subscriber identity module) in the MTC device for security and proper authentication with the network.

2.2 Challenges of Cellular-enabled Massive MTC

Cellular networks have been engineered and optimized to carry increasing amounts of mobile data, but when it comes to enabling massive MTC over the cellular network, a set of novel technical challenges needs to addresses as follows.

- **Deployment-related challenges** including massive density, arbitrary locations, shadowed indoor and high mobility; opportunities include use cases with static fixed locations.
- **Traffic-related challenges** including continuous uploading, critical real-time uplink and/or downlink connections, and very large message size; opportunities include use cases with (very) delay-tolerant traffic.
- **Energy-related challenges** including ultra-low energy consumption and high energy efficiency; opportunities include possibly automated battery charging.
- **Congestion-related challenges** including low overhead and scalable signaling with deep sleep modes, yet timely paging or network-initiated device triggering for time-critical downlink connections.

- **Complexity-related challenges** including transmitter/receiver design requirements, computational/storage capabilities, and device longevity through backward compatibility, to maintain functionality over a long period even as cellular technologies evolve.

Let us consider the furthermore details on the above technical challenges and reasons behind them by considering differences between Human Type Communication (HTC) and Machine Type Communication (MTC) in the next sub-section.

2.2.1 Differences between HTC and MTC over cellular networks

The requirements to support MTC services over cellular are inherently different and more diverse than traditional Human Type Communication (MTC) services. MTC services add several challenging new constraints to the range of service requirements in cellular networks, as summarized in Table 2.3.

Table 2.3 Differences between HTC and MTC over cellular networks [ref]

Service Requirements	HTC over cellular	MTC over cellular
Traffic Direction	Mostly downlink; although uplink traffic is increasing over the last years due to interactive applications such as social networking, humans still download more than they upload.	Mainly uplink data to report sensed information. For some applications, symmetric uplink and downlink capacity are needed to allow for the dynamic interaction between sensors and servers.
Message Size	The size of the messages is generally big, motivated by demanding applications such as multimedia and real-time transmissions, including video streaming.	The size of the messages is generally very short (e.g. very few bits of the reading of a meter, or even just 1 bit to inform of the existence or absence of a given event).
Connection and Access Delay	Human-based applications tend to be very demanding once a connection has been established. However, although not desirable, longer connection delays are typically well-tolerated.	Many M2M applications will be based on duty-cycling, i.e., having devices sleeping and just waking up from time to time to transmit data. For some applications, the connection delays should be very short to ensure quick access to the network when waken up.
Transmission Periodicity	Human-based data traffic is very random and asynchronous in nature. Also, the frequent transmission of control information is required to ensure high throughput and good delay performance.	Very wide range of alternatives. For many applications, transmissions will be very sparse in time. Also, many applications will have known periodic patterns (e.g. programmed tasks).
Mobility	Mobility management and exchange of location information are constantly required to ensure seamless connectivity and allow for roaming.	For many M2M applications, mobility is not a major concern. Some applications may have no mobility at all.
Information Priority	In general, there is no major differentiation between users in terms of priority, but only between applications for each user.	Some M2M applications may transmit critical information and thus require very high priority with a detailed level of granularity.
Amount of Devices	Lower than in M2M. At most, hundreds of devices per connection point. Typically, tens of devices per connection point.	Higher than in human-based communications. Hundreds or thousands of devices per connection point.
Security and Monitoring	Humans can raise an alert in the case of troubleshooting or tampering.	M2M devices cannot raise an alert in the case of malfunctioning or tampering.
Lifetime and Energy Efficiency	Although annoying, humans can recharge batteries in a daily manner.	Once an M2M network has been deployed, some devices may require to operate for years or decades without maintenance.

2.2.1 Congestion-related challenges

Congestion may occur due to simultaneous signaling messages from a massive number of MTC devices in the network. This can be significant as they may lead to peak load situations and may have a tremendous impact on the operations of a mobile network. Mainly on the performance of vital nodes with scarce resources such as control plane nodes (as examples SGSN, GGSN in UMTS and MME, PDN GWs in EPS), and radio access network (RAN) (as examples eNB, base stations, and RNC) can be harmfully impacted due to these signaling congestions.

As the regulatory body 3GPP, focuses on finding efficient mechanisms to handle congestion mainly at the core network and radio network parts of the cellular networks that may happen due to nearly simultaneous signaling messages issued by a significant number of MTC devices. Thanks to the wide bandwidth of LTE and other emerging optical fiber technologies its highly unlikely to occur congestion due to user data packets in radio network and control plane of EPS.

2.2.2 Bulk MTC Signaling Handling

In the context of MTC, signaling congestion may happen due to massive attempts from a potential number of MTC devices to attach and connect to the network all at once. Under this type of condition core network nodes that responsible for authentication and tracking area updates may be running out of computation power to respond for these all the signaling coming from MTC devices ***[3GPP, "Service Requirements for Machine-Type Communications," TS 22.368 V10.1.0, June 2010].***

Therefore, there is a requirement of a mechanism to handles a bulk of similar signaling messages from MTCs in a single shot. In this case, it is possible to mitigate signaling congestion by handling signaling messages, common to a number of MTC devices, by means of bulk processing. Figure 2.5 depicts an example whereby Non-Access Stratum (*NAS*) signaling messages from different MTC devices can be aggregated for bulk handling in the network. Effectively, in case a large number of MTC devices are simultaneously triggered to send a NAS signaling message (as an example TAU request, massive attach requests) to eNodeB. In such situations, NAS signaling messages can hold back for a pre-defined timeout or until a number of NAS signaling messages arrive to proceed with a bulk of NAS signaling messages toward the MME.

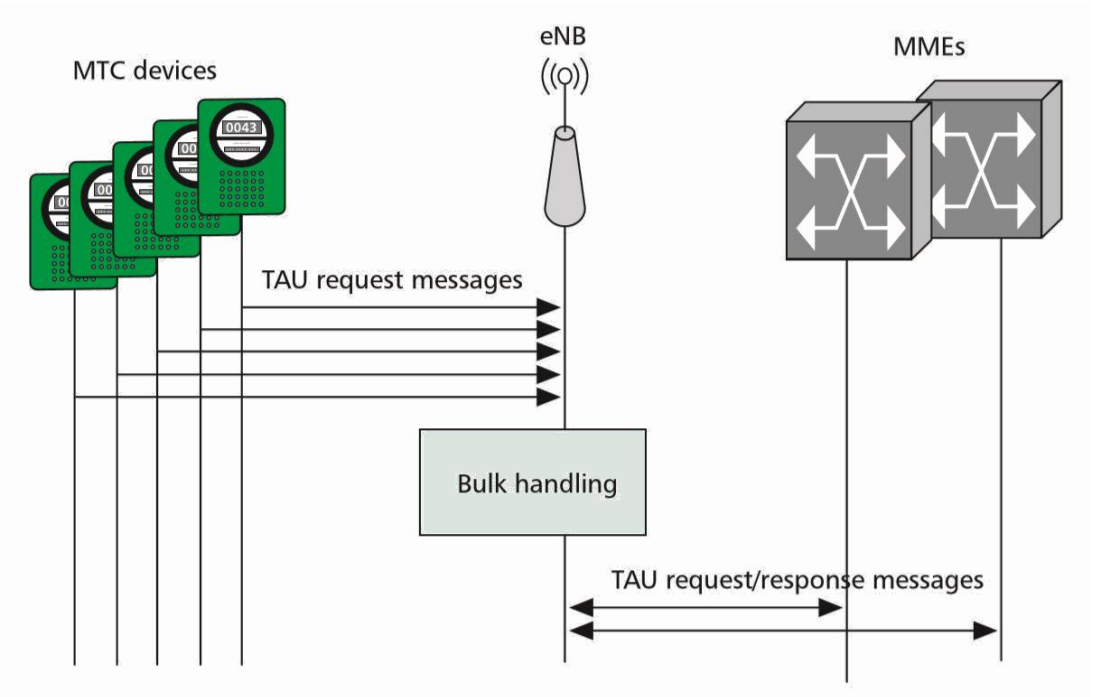


Figure 2.5 NAS signaling messages can be aggregated for bulk handling

2.2.3 Random Access Procedure

When considering the Random Access (RA) procedure in LTE as the first step of proposing solutions for signaling congestions at radio network due to massive numbers of RA signaling initiate at ones form MTC device of the network. RA signaling considers as the first step of establishing an uplink direction connection between any user equipment (UE) and a base station (BS). According to the first step of the RA process BS responsible for broadcasting information about available random access channels (RACH). When UE needs an uplink connection to transmit data, each UE selects randomly one of available RACH to send a scheduling request ***[S. Ali, N. Rajatheva and W. Saad, "Fast Uplink Grant for Machine-Type Communications: Challenges and Opportunities", IEEE Communications Magazine].***

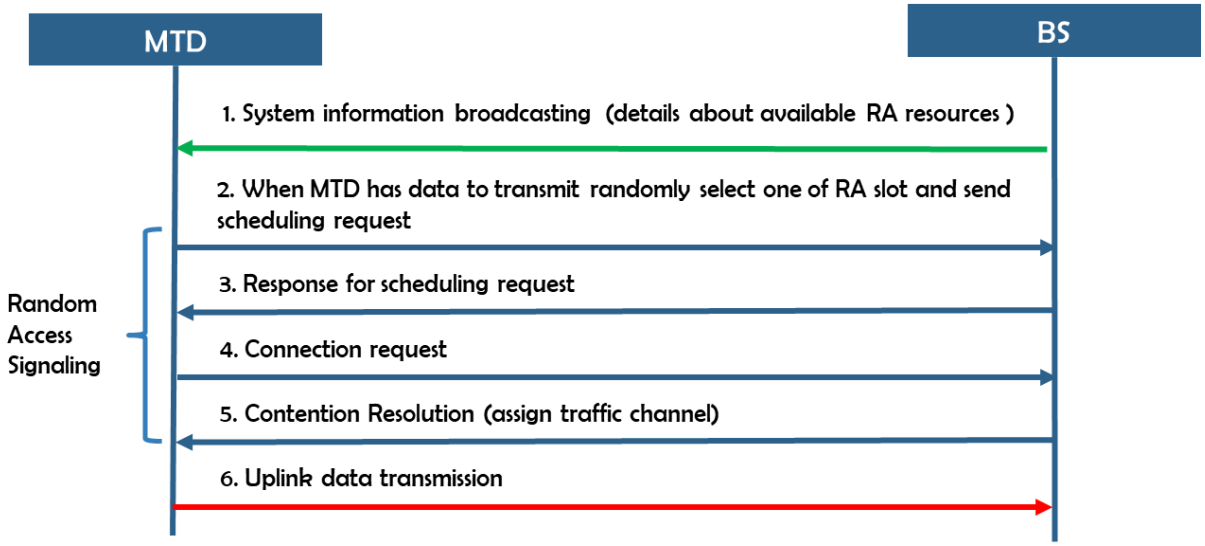


Figure 2.6 Random Access (RA) Procedure

RA process exhibits an inability to support massive cellular-enabled MTC due to collisions, delays and signaling overhead problems associated with the RA process. When the number of MTDs competing for RA is larger than the number of available RA slots, the probability of RA scheduling request collisions will be increase. When more than one MTD selects the same RA slot collisions will occur and subsequently BS will attempt to decode collided request and send responses. However, if the BS unable to decode the collided requests, demanding MTDs have to re-send the request and ultimately this will case for longer delays. Even so back off mechanisms can reduce collisions it will increase the delays. Likewise, RA signaling itself takes time to establish uplink transmission, which exacerbates delay problems. Moreover increasing the number of RA slots not feasible since the physical uplink channel has to share for the RA process and uplink transmission. When signaling overhead of the RA process is larger than the actual packet size that intended to transmit, the RA process is highly ineffective for cellular-enabled massive MTC ***[S. Ali, N. Rajatheva and W. Saad, "Fast Uplink Grant for Machine-Type Communications: Challenges and Opportunities", IEEE Communications Magazine].***

Uncoordinated transmission systems entirely depend on the random access of uplink-direction transmission resources. Hence, MTD selects a randomly uplink channel and transmits data by totally eliminating signaling overhead however when the number of devices exceeds the number of uplink channels this type of transmission system extremely get suffer by collisions.

2.3 Fast Uplink Grant for MTC

The requirement of novel wireless medium access schemes for cellular-enabled MTC has arisen due to the congestion at the radio access network. To address this requirement the concept of Fast Uplink Grant has proposed by 3GPP. Fast Uplink Grant enables BS to accurately predict MTDs, which has data to transmit and actively allocate uplink resources to MTDs using one broadcast signal for the entire cell. Hence, whenever MTD has data packets to transmit, it can wait a pre-defined time interval until BS allocates a fast uplink grant without immediately sending an RA request. Anyhow, if Fast Uplink Grant doesn't receive during that time interval, MTD decides to send an RA request to the BS by following a traditional way. Figure 2.7 illustrates this decision making process at the MTD.

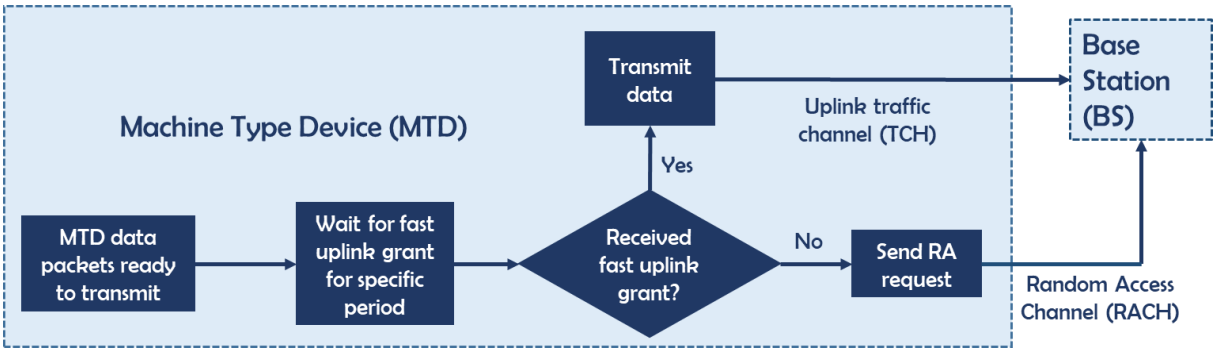


Figure 2.7 RA decision-making process at MTD

Since Fast Uplink Grant itself doesn't require any signaling part, it can reduce the overall RA request amount of the network and prevent signaling congestion situations. The amount of RA request cut completely depends on the source traffic prediction capabilities of the BS. Furthermore, Fast Uplink Grant can to satisfy a Quality of Service (QoS) requirements of MTDs by a prioritized allocation of uplink radio resources for MTDs with stricter latency requirements. Figure 2.8 graphically illustrated the Fast Uplink Grant procedure briefly.

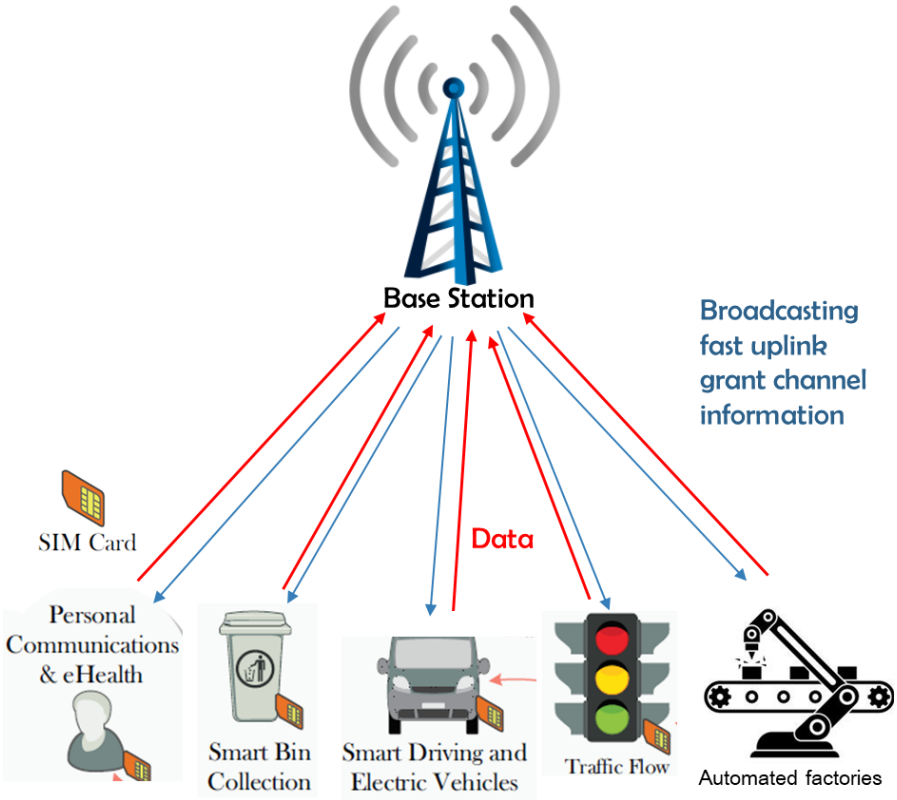


Figure 2.8 Fast Uplink Grant Procedure

Table 2.4, describe how fast uplink grant overcome major three problems of the RA process and uncoordinated transmission systems.

Table 2.4 fast uplink grant as a balanced approach between random access and uncoordinated access

	Coordinated (Random Access)	Fast Uplink Grant	Uncoordinated (Random Data Transmission)
Signaling	High (6 resource blocks (RB) + messages 2 to 4)	Small (one broadcast message for entire cell)	Zero
Collisions	High (number of MTDs >> number of RA slots)	Zero	High(number of MTDs >> number of RBs)
Latency	Waiting for RA slot + RA signaling	Small (a few ms if the grant is allocated on time)	High (when the number of MTDs >> number of RBs)

Briefly, Fast Uplink Grant is an intelligent wireless-media access technology specifically tailored for cellular-enabled massive MTC. It has the capability for optimal allocation of uplink-direction radio resources by predicting source traffic and considering QoS Requirements of MTDs.

2.3.1 Challenges of Fast Uplink Grant

The first drawback of the fast uplink grant is the possibility of wasting resources whenever an MTD that receives a fast uplink grant does not have data to transmit [7]. Moreover, if the fast uplink grant is not received within the maximum tolerable access delay of the data packets, packets will be dropped, yielding transmission failures. This can potentially lead to unreliable and high latency MTD transmissions. Hence, to adopt the fast uplink grant for MTC, one of the main challenges that must be overcome is the optimal selection of MTDs by the BS. This MTD selection process, in turn, faces two key challenges. First, there is a need to predict the set of MTDs that will likely have data to transmit at any given time. By doing accurate predictions, the BS can solve the problem of allocating the fast uplink grant to silent MTDs. Once predictions are properly implemented, the BS must also be able to determine the scheduling sequence of MTDs. This challenge is particularly pronounced when the number of devices significantly exceeds the number of resources. Hence, sophisticated scheduling algorithms are needed to enable fast uplink grant allocation. Next, let us consider the proposed a two-stage approach for enabling the fast uplink grant for MTCs.

2.3.2 Source traffic predication

In the Fast Uplink Grant process, the allocation of resources is performed by the base station. However, with the absence of any scheduling request or any sign of readiness from the device side, the base station alone wouldn't have a clue to allocate resources efficiently. For example, if the base station randomly picked devices to allocate resources, there can be situations where the base station allocates resources to MTDs that are not ready to transmit. On the other hand, another MTD could be waiting to transmit data but uplink resources might not have been available. Ultimately this results in wastage of resources. To overcome this, BS must use a sophisticated method of source traffic prediction. Here we need to understand that a traditional traffic prediction model such as aggregate traffic prediction does not address this particular requirement. This type of traffic modeling only predicts the total number of devices or packets entering the system at any given time. On the contrary, here we need to identify the devices from their IDs which are waiting to send data to the network. The information about the summation of the MTD data packets would not sufficient to allocate uplink resources by the BS. The BS needs to know what exact MTD is going to enter the transmission at any given time step beforehand. To tackle this problem, we need to identify the anticipated types of traffic. There can be two main types of MTD transmissions. Periodic reporting and event-driven transmissions. Thus two

types of corresponding traffic can be expected in the network. Each of these types is explained in the next sections.

2.3.3 Periodic traffic

In periodic traffic, MTDs transmit measured sensory data packets from the observations of the physical environment at specific, predetermined times. As an example, consider a temperature sensor in a smart refrigerator that sends measurement results every day at 11:55h. In this special situation, duration of the periodic interval is 24 hours. In reality, the duration of these periodic intervals could be as low as a few milliseconds and up to once a month. However, in a practical network collecting information about these periodic traffic patterns within the MTC application level is not unrealistic since this periodic intervals are possible to change over time. Therefore, BS has to learn about these transmission patterns from previously recorded MTD transmissions history using advanced time-series pattern mining methods. Existing literature proposes techniques such as calendar-based periodic pattern mining, sequential association rule and non-homogeneous Poisson process (NHPP). Unfortunately, as many of these algorithms based on complex mathematical functions, they have limited pattern-mining capabilities. Furthermore when it comes to MTD networks with thousands of MTDs that transmit various data sizes in various period durations become extremely complex and consume plenty of processing power. Therefore it is possible to take an approach to use machine learning-based time-series pattern recognition techniques to address those problems. Especially existing literature lack any implementation associate with the periodic traffic predictions and this study was carried out to fulfill this shortcoming.

2.3.4 Event driven traffic

In event-driven traffic, whenever a special MTC event or IoT event causes a vast number of MTDs that detect the same event must initiate data transmission to the BS. When one of MTD is triggered by an event, how to identify the other set of MTDs that experience the same event became the major challenge of event-driven traffic prediction. The prior research has tried to exploit the correlation for source traffic prediction during MTC events. This can be simplified as upon detection of a certain MTC event, the network can predict which other MTDs will experience the same event and are likely to start transmitting. This can be utilized to enable Fast Uplink Grant for MTD that exhibits event-driven transmissions.

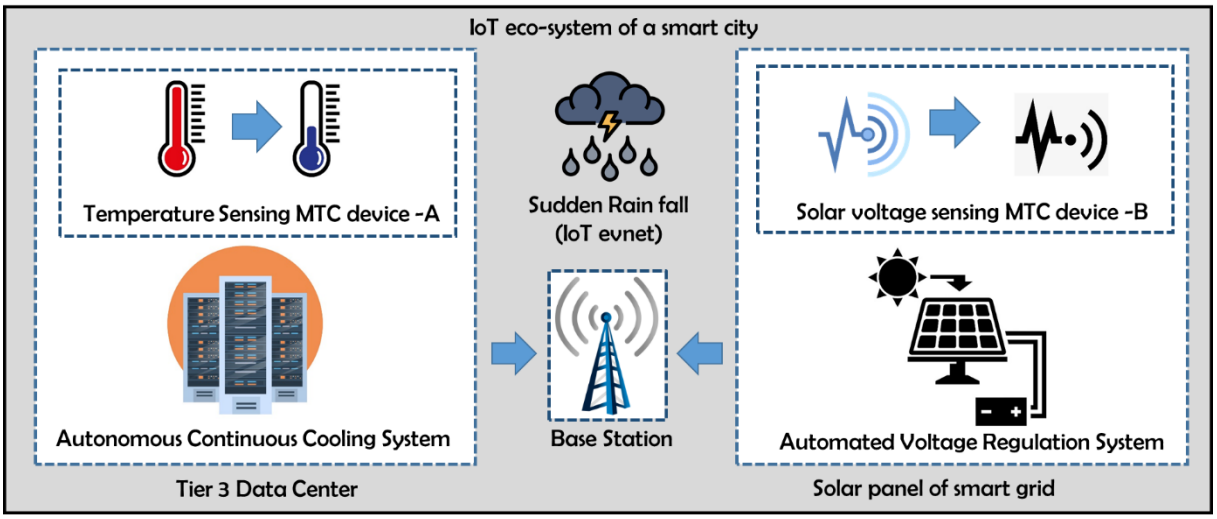


Figure 2.9 Example for event-driven transmission within MTC ecosystem of a smart city

Consider a temperature sensing MTC device and solar voltage sensing MTD illustrated in figure 2.9 as a sample use case of event-driven transmission, where two MTC devices belong to complete two different MTC applications. Then consider a sudden rain falling situation as an MTC event that

experience by both MTDs. Especially when considering MTC ecosystem of a smart city with high device density there will be more independent MTC applications that get the effect by sudden temperature chance and humidity change in a worm day due to rainfall. The environmental changes due to sudden rainfall will case many MTDs to start transmitting data packets.

If the MTC network operator can collect the information about a set of MTDs effect by rainfall from a subscription in the HSS this task is rather trivial. However, in practice, due to privacy and security issues, as well as the financial benefits of data for the application, the MTDs might not reveal the nature of the application to the network operator. Moreover, the set of MTDs effect by different MTC events might change at different times, due to changes in the environment and the MTC applications itself.

Therefore, BS has to separately identify the set of MTC devices that get the effect by the same event from the transmission history of the MTC network as a whole. However, enabling Fast Uplink Grant for MTC network with event-driven transmissions is generally possible due to the following reasons.

- Transmission history of each MTD can be model into a binary time series
- History of transmission of each MTD can be extracted from the history of random access requests
- Typically MTC events propagate over a geographical area and time domain
- Random access technologies are not completely neglected
- Ones the periodic traffic patterns learned, periodic transmissions will not initiate RA requests
- Therefore still receiving RA requests can be considered as an event trigger
- Furthermore, RA can be used to detect high entropy transmissions, unusual and random traffic generated by MTDs

For event-driven traffic, the existing literature focuses on using Directed Information (DI) to identify the correlation between transmission from different MTDs **[Samad DI paper].** The concept of Directed Information is a powerful tool to measure the causality and the information flow between sequences of random variables [8]. But there are some critical shortcomings in this method.

2.3.5 Optimal allocations

Once the set of MTDs that have data to transmit is predicted, if the number of devices is smaller than the number of available uplink channels, all the MTDs can be scheduled to transmit. Typically wireless frequency bands are considered as limited transmission resources. Therefore, when the number of MTDs exceeds the number of available uplink channels at any given time, the network must select which MTDs can be granted access. Especially when considering an MTC network with a high MTD density number of MTDs competing for the uplink channel will be much higher than the number of available uplink channels. Because of that MTDs should be scheduled based on their latency requirements such as the maximum tolerable delay. A maximum tolerable delay is the maximum time amount that MTD can wait until it receives the uplink channel without harmfully affecting the application performance. If fast uplink grants are allocated randomly, the delay-sensitive MTC application may suffer from not receiving the fast uplink grant at a time.

If the BS has full knowledge of the QoS requirements of all MTDs, this scheduling can be performed in a centralized manner. However, in a realistic scenario, such information might not be available to the BS and any fast uplink grant allocation algorithm should be able to select MTDs in an uncertain environment. Therefore, the design of sophisticated algorithms for optimal fast uplink grant allocation is needed. Here, researchers present some initial directions toward building such scheduling algorithms that exploit recent advances in machine learning and Reinforcement Learning (RL) to optimize the allocation of fast uplink grants to MTDs. Each MTD should be allowed to transmit in a given frequency band until they finish their transmission. This can help in dealing with various data packet sizes of MTC applications.

2.4 Directed Information

Directed information (DI) is a measure of information theory and it measures the amount of information that flows from one process to another process. The term directed information was introduced by James Massey and described as conditional mutual information. In probability theory and information theory, conditional mutual information is a basic form of the expected value of mutual information between two random variables. Therefore, DI is a powerful tool to investigate the causality and flow of information between sequences of random variables.

The general equation of directed information give as follows,

$$I(X^n \rightarrow Y^n) = \sum_{i=1}^{n} I(X^i; Y^i | Y^{i-1})$$

Where,

$I(X^i; Y^i | Y^{i-1}) =$ Mutual information between X^i and Y^i conditioned on Y^{i-1}

$X^N = \{X_1, X_2, X_3 \dots X_N\} = Length\ N\ sequence\ of\ Random\ variables$

$X^i =$ Element i of X^N

The algorithms proposed in the paper relies on calculating DI between sequences of the length of two. The Directed Information for such sequences, $\{X_kX_{k+1}, Y_kY_{k+1}\}$, is given by,

$$I(X_k^{k+1} \rightarrow Y_k^{k+1}) = H(X_k) - H(X_kY_k) + H(X_kX_{k+1}Y_k) + H(Y_kY_{k+1}) - H(X_kY_{k+1}Y_kY_{k+1})$$

Where,

$$H(X_1, \dots, X_n) = -\sum_{x_1, \dots, x_n} p(x_1, \dots, x_n) \log p(x_1, \dots, x_n)$$

The entropy terms on the right-hand side are calculated and the Directed Information is derived in the DI paper **[Samad DI paper].** A novel method for causal inference is based on the concept of directed information **[Universal Estimation of Directed Information]**, which can be used to infer causality between sequences of random variables. Considering two sequences of random variables, past and present values of the first sequence and past values of the second sequence can be used to evaluate the present value of the second sequence. Therefore, this directed information used as a powerful concept in predicting seizures in epilepsy patients and causality between neurons of the human brain **[Universal Estimation of Directed Information]**.

2.4.1 Source traffic prediction with directed information

According to **[Samad DI paper]** researchers have presented a novel approach to predict the source traffic in event-driven MTC. First, they have explored how MTD transmissions can be correlated due to certain relationships between IoT events. Then, they have used a binary sequence to model the transmission history of different MTDs. The history of the RA requests from each MTD can be considered as the transmission history where researchers have proposed to model that transmission history with a sequence of binary random variables, where transmitting at each time is presented by 1 and being silent with 0.

By introducing the concept of DI, they have proved how it can infer causality between different sequences. Then a new algorithm has developed to predict the set of MTDs that have data to transmit.

Proposed Directed Information algorithm for event driven traffic prediction [DI paper]

1: Fix the length of events to the maximum length L among events.
2: For every time step that event had happened, set the value to 1 and 0 otherwise.
3: ***for*** *every pair of MTDs X and Y do:*
4: ***for*** *every pair XkXk+1 and YkYk+1, k ∈{1, ...,L − 1} and XkXk+1 and Yk+iYk+i+1, i ∈{1, ..,L − k} do:*
5: Calculate the probability distributions (5) for entropies (6).
6: Calculate DI between from MTD X to MTD Y and vice versa (6).
7: Create the set K for every MTD, such that I(XkXk+1 → Yk+iYk+i+1) 6= 0, and include i and DI.
8: ***end***
9: ***end***

2.4.2 MTD Network Simulation

Artificial data has been generate to represent the transmission patterns of the MTDs. Four MTDs were considered, X, Y, Z, and T. The length of the MTC event is considered to be 12 time steps. For MTD X, the event happens at times t ∈ [1, 2, 3, 4, 7, 8, 9]. In other words, at those times, there can be a transmission, and at other times, MTD X is silent. For MTD Y, we consider t ∈ [4, 5, 6, 8, 9, 10, 11]. For MTD Z, event occurs 3 time steps after X with probability 80% and, for MTD T, the event happens 2 time steps after X with probability 20%. Couple of example transmission pattern of MTD X, Y, Z and T can be represented as follows.

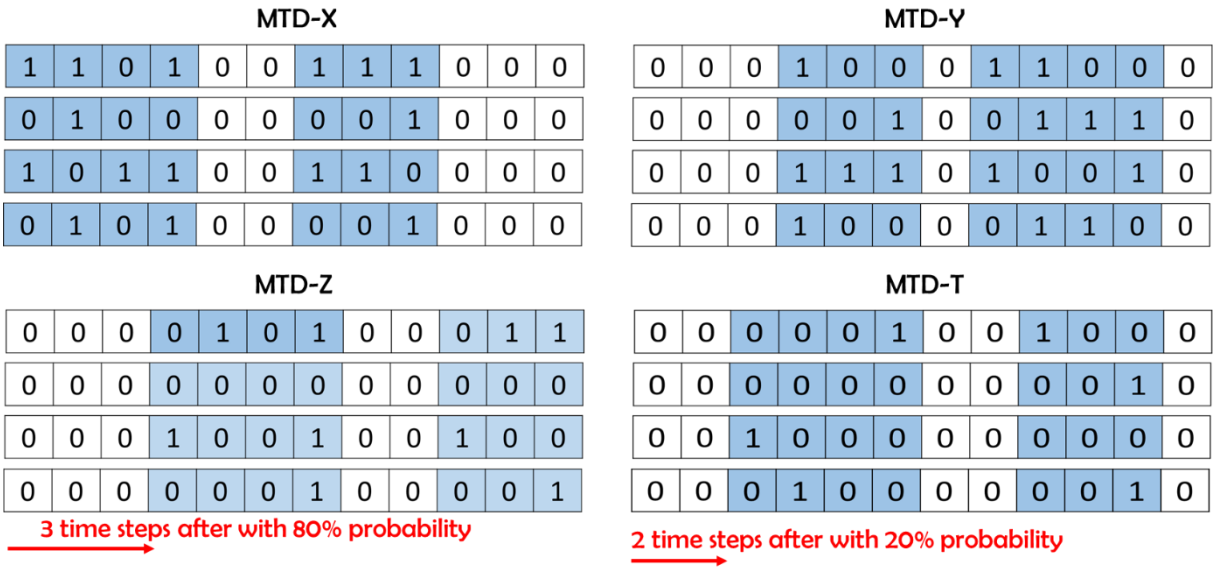


Figure 2.10 Example transmission patterns for MTD X, Y, Z, and T

2.4.3 Simulation Results

According to the results are presented in **[Figure 2.11]**,

$$I(X_{k,k+1} \rightarrow Y_{k+i-1,k+i})$$

is presented in element (i,k) of the matrix of the results directed information for X MTD transmission to Y MTD transmissions.

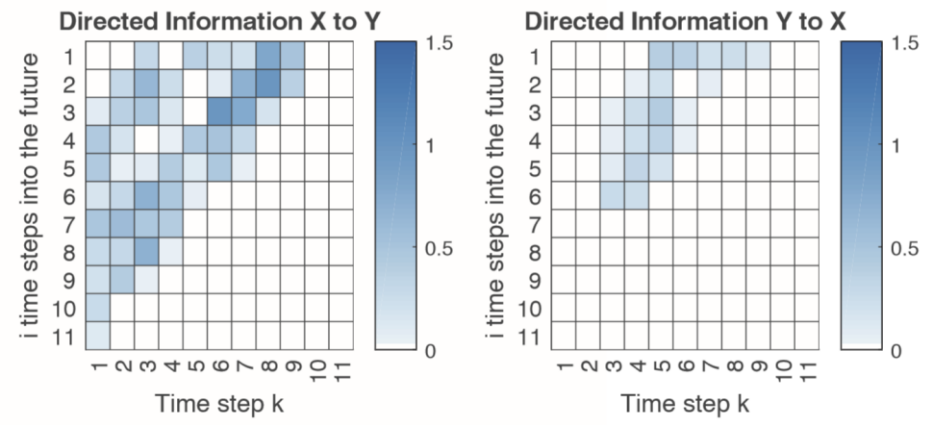


Figure 2.11 Directed Information flow between MTD X and MTD Y transmissions [Samad DI paper]

For the results, it is clear that DI has the quality of identifying event propagation direction, but it cannot generate a solid prediction of the sequence due to its nature of the output. Thus, the scalability of DI learning method is limited. Furthermore, the DI algorithm takes only two MTDs for consideration at a time. Without data preparation calculations, this method has a complexity of O(n2) with the number of devices. In addition, this complexity is only for finding the direction of event propagation. A separate algorithm should be used to generate stochastic prediction rendering this method further complex.

2.5 Reinforcement Learning

Reinforcement learning is an area of machine learning which concerned about how software agents learn by experience to make decisions in an environment to maximize some notion of cumulative reward.

Existing research has used a reinforcement learning (RL) technique called Multi-Armed Bandits (MAB) with the Upper Confidence Bound (UCB) algorithm for optimal allocation of Fast uplink grant **[MAB paper].** MAB is a class of RL problems that deal with decision making in uncertain environments with limited or no prior information **[RL book]**.

The basic MAB problem consists of a set of arms (available actions) that can be chosen by a decision-making agent. The agent plays an arm at each time and receives a reward. The rewards are drawn from an unknown probability distribution. The agent has no prior information about the rewards of each arm and it has to randomly select arms at the start. However, after some time with observing the rewards, it tries to find the best possible arm based on the expected reward of each arm. The objective of an agent to maximize the cumulative reward over time. In MAB, the notion of regret defined as the reward difference between the best possible arm and selected arm.

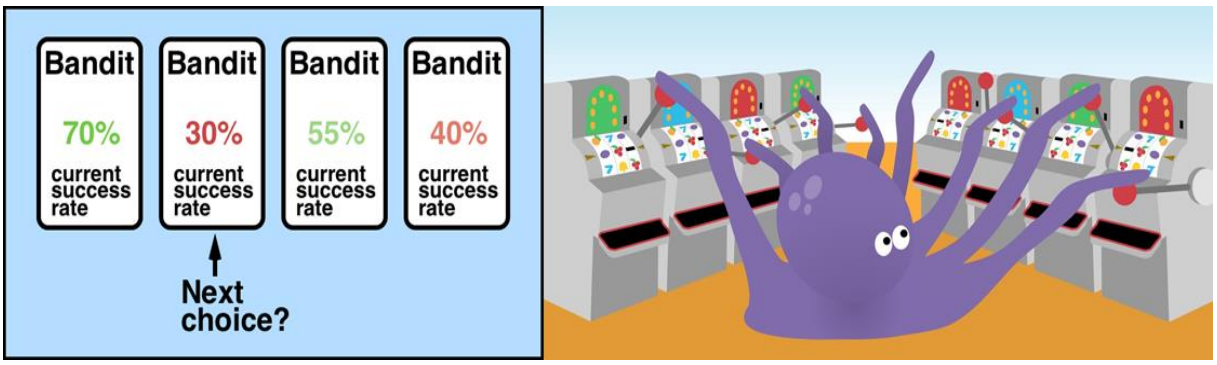


Figure 2.12 Nature of a Multi-Armed Bandit problem

2.5.1 Quality of Service (QoS) based Reward Function

The optimal allocation fast uplink grant for MTDs needs to be performed by considering the Quality of service (QoS) requirements of MTDs. First researchers have defined a QoS metric as a combination of the value of data packets, maximum tolerable access delay, and data rate.

The value of the information at a given time for each MTD has determined using a relative pairwise comparison of all IoT applications [9]. A structuring technique called Analytic Hierarchy Process has used to calculate the important weight for each packet as a percentage.

The maximum tolerable access delay [$d_i(t)$] has defined as the total number of time steps that can be tolerated from the time instance, in which the data packet is ready to be transmitted. In simple terms, maximum tolerable delay values represent the latency requirements of the MTD. **[Figure 2.13]** graphically represents the MTC application has a latency requirement between 100ms to 1000ms.

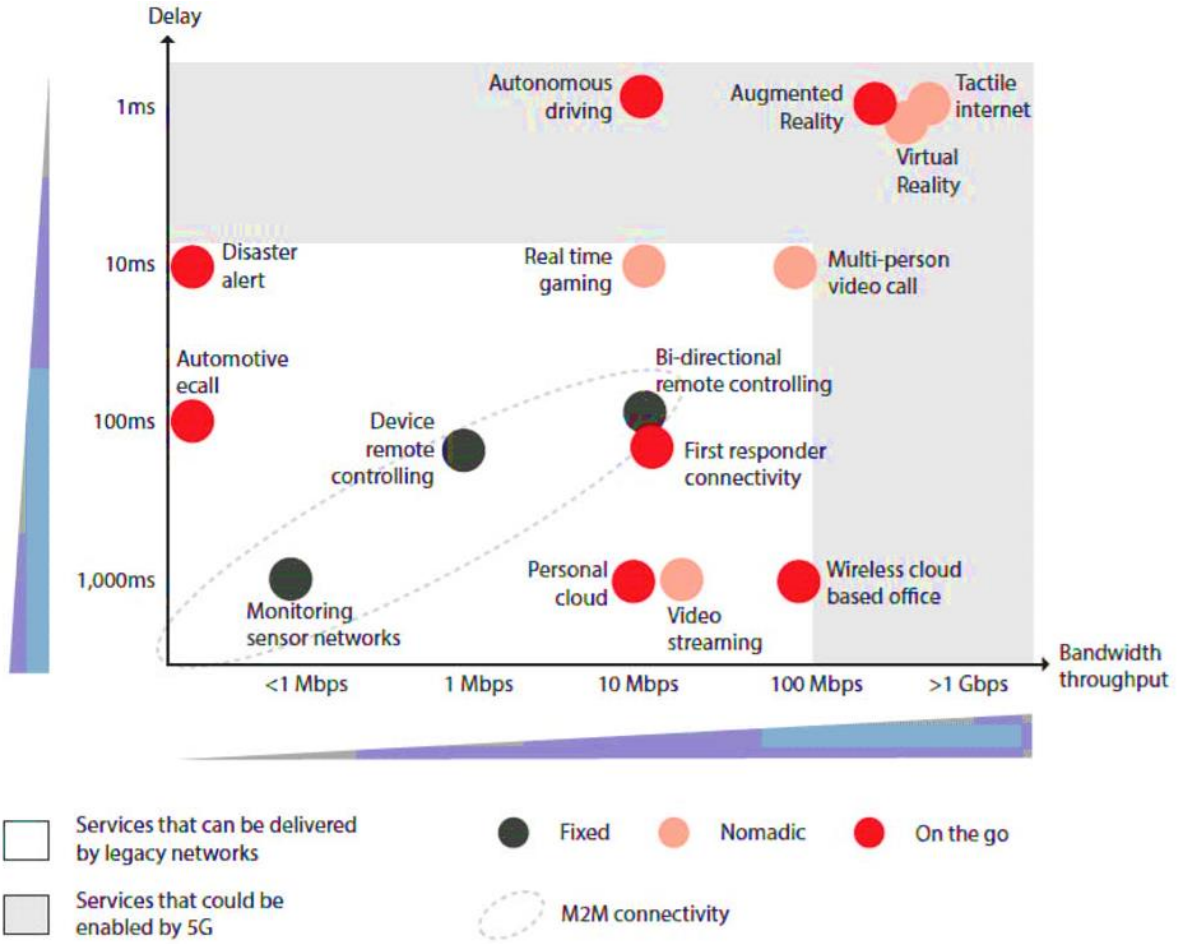


Figure 2.13 Bandwidth and latency requirements for MTC applications

The normalization of maximum tolerable access delay has achieved by mapping $d_i(t)$ into a number between 0 to 1 using modified Sigmoid function as follows,

$$f[d_i(t)] = a - a^{-be^{-cd_i(t)}}$$

Where a, b and c parameters define the characteristics of Sigmoid function. The function gives large values for smaller $d_i(t)$ values by allowing delay-sensitive applications to have higher utility function values. Furthermore, transmission and propagation delays are assumed to be already subtracted from the total latency requirements of each IoT application.

The normalized rata rate can define as,

$$C_i^n(t) = \frac{C_i(t)}{C_{max}}$$

$C_i(t) = achieved\ throughput\ by\ the\ link\ between\ MTD\ and\ BS$
$C_{max} = maximum\ data\ rate\ can\ be\ achieved\ by\ the\ node\ that\ has\ the\ best\ channel\ to\ the\ BS$

Then they have considered utility function based on the above QoS metrics as represented in the following equation.

$$U_i(t) = \alpha V_i(t) + \beta C_i^n(t) + \gamma f[d_i(t)]$$

$V_i(t) = value\ of\ data\ packet \in [0,1]$
$C_i^n(t) = normalized\ throughput\ of\ the\ link\ between\ MTD\ and\ BS \in [0,1]$
$f[d_i(t)] = normalized\ maximum\ access\ tolerable\ delay \in [0,1]$

$\alpha + \beta + \gamma = 1$ is weight parameters that can use to modify the effect of each QoS parameter on the utility function value. However, in a practical network, it is unrealistic to have information about QoS parameters along with HSS subscription details of MTC application due to the following reasons.

- Due to privacy, security and financial benefits consideration of different MTC applications, QoS parameter wouldn't reveal to the MTC network operator.
- Achieved throughput of the system can be changed stochastically due to charges of channel state information (CSI) of the MTDs
- The maximum tolerable delay of MTDs might stochastically change due to changes in propagation delays, queuing delays and MTD processing delays
- After MTD packets are queued at BS tolerable delay of that data packets decrease with the time

Therefore, utility function values have a stochastic probability distribution.

2.5.2 Optimal Fast Uplink Grant allocation with MAB theory

BS has to learn the expected values of the utility function for each MTD over time. To achieve these researchers has use multi-armed bandit (MAB) theory. According to the proposed MAB framework in **[figure2.14],** player or decision maker (BS) pulls arms or actions (select MTDs for fast uplink grant allocation) from the available set of arms (active MTDs). Each allocation generates a reward (utility function value) after being played, based on a distribution that BS not known.

$$\boldsymbol{\theta_i(t) = I[d_i(t) > t_i - t_s].\mathbf{1}[C_i(t) > \rho].U_i(t)}$$

$I = 0\ ; C_i(t) < \rho$ $\quad$ $\boldsymbol{data\ transmission\ unsuccessful}$
$I = 1; C_i(t) > \rho$

$I = 0\ ; d_i(t) < t_i - t_s$ $\quad$ $\boldsymbol{data\ packets\ will\ drop}$
$I = 1; d_i(t) > t_i - t_s$

$\theta_i(t) = Reward\ of\ MTD\ i\ at\ time\ step\ t$
$\rho = data\ rate\ threshold\ required\ for\ data\ transmission$
$t_i = time\ step\ that\ fast\ uplink\ grant\ received$
$t_s = time\ step\ that\ MTD\ had\ ready\ to\ transmit\ data$

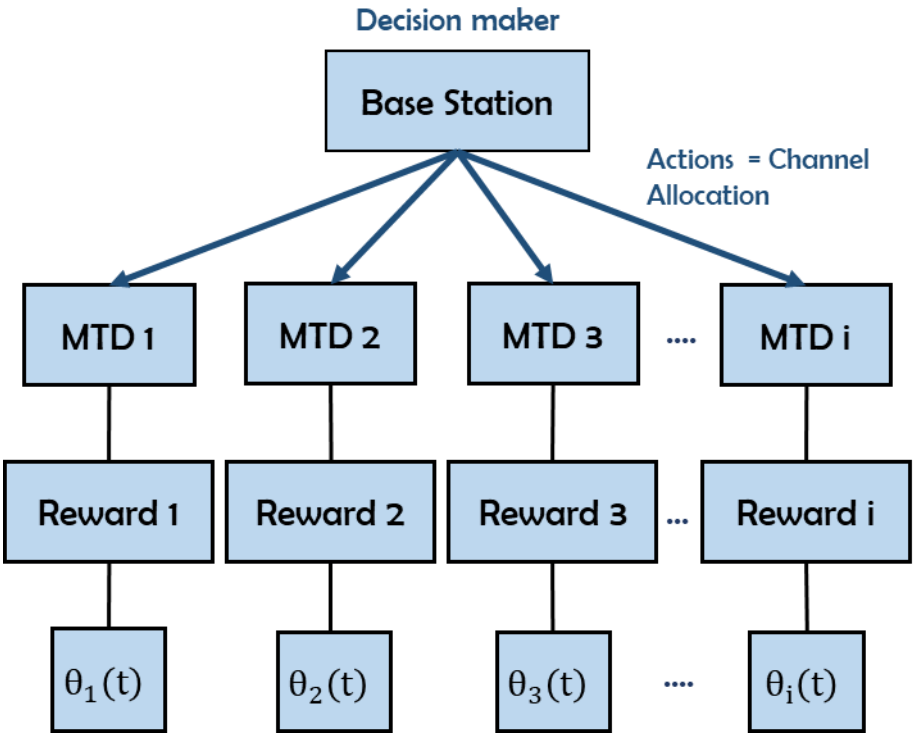


Figure 2.14 Proposed MAB framework for Optimal Grant Allocation

2.5.3 Upper Confidence Bound Algorithm

BS only observe the reward of the selected MTD and MAB algorithm's aim is to maximize the cumulative reward over time. This can achieve by often selecting of the MTDs that has higher reward and that called exploitation of arms. However, MAB algorithm should explore all the other arms enough times to discover their expected values more precisely and that called the exploration of arms. In order to achieve the trade-off balance between exploitation and exploration upper confidence bound (UCB) algorithm [10] has been used.

Step 01:- At each time step t UCB algorithm estimates two parameters for all MTDs in the network,

$n_i(t) = Number\ of\ times\ MTD\ i\ has\ been\ selected\ up\ to\ time\ step\ t$
$R_i(t) = Sum\ of\ rewards\ of\ the\ MTD\ i\ up\ to\ time\ step\ t$

Step 02:- Then the UCB algorithm computes the average reward of MTDs up to time step t,

$$\bar{r}_i(t) = \frac{R_i(t)}{n_i(t)}$$

Step 03:- Compute the confidence bound of MTDs at time step t,

$$\Delta_i(n) = \sqrt{\frac{s\log(t)}{n_i(t)}}$$

Step 04:- Compute the confidence interval of MTDs at time step t,

$$[\bar{r}_i(t) - \Delta_i(n)\ , \bar{r}_i(t) + \Delta_i(n)]$$

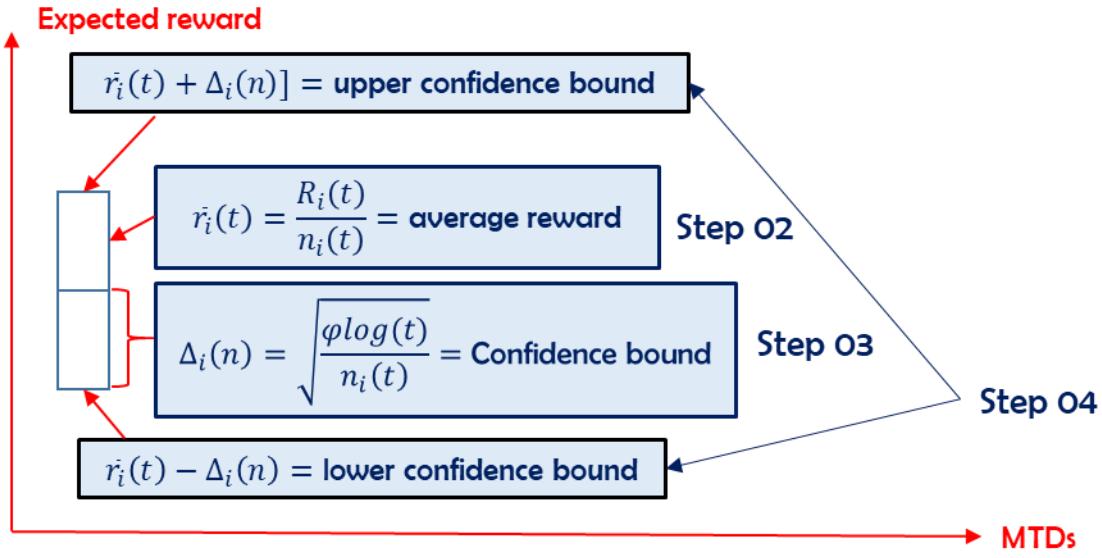


Figure 2.15 Steps of Upper Confidence Bound algorithm

Step 05:- Select the MTDs with higher upper confidence bound value,

$$[\bar{r_i}(t) + \Delta_i(n)]_{max} = \left[\frac{R_i(t)}{n_i(t)} + \sqrt{\frac{slog(t)}{n_i(t)}}\right]_{max}$$

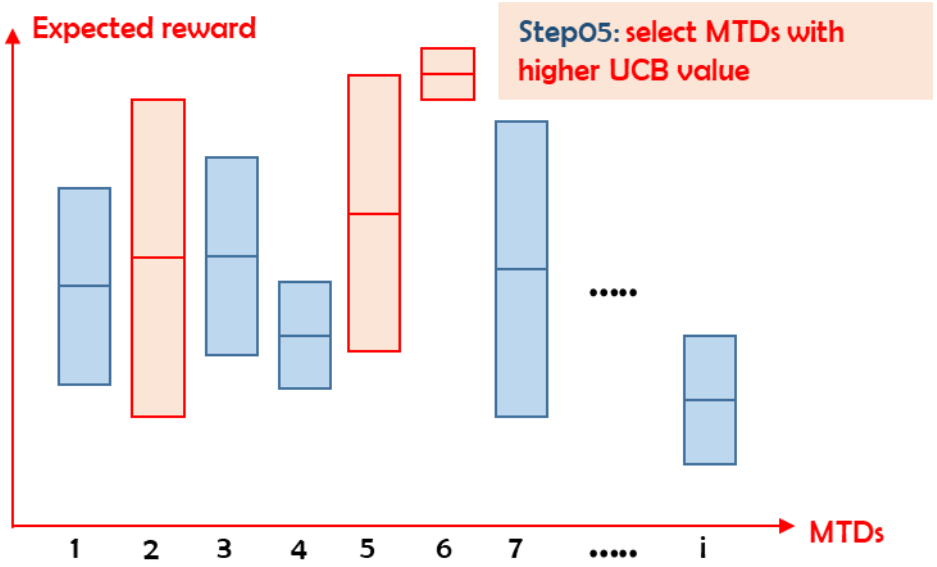


Figure 2.16 MTDs with higher UCB value

2.5.4 Probabilistic MTD Availability

In classical MAB problem all the arms available to the player at all the time instants. However, for this Fast Uplink Grant scheduling problem a set of available arms various over time because after each MTD transmission they are silent for some time. This type of special MAB problem called as sleeping Multi-Armed Bandit problem.

Therefore, as illustrated in **[figure 2.17]** silent MTDs term as sleeping MTDs and MTDs that have data packets term as active mTDs. Researchers have assumed BS has a prediction algorithm to determine the set of MTDs with a certain probability. Therefore, the availability of data packets are probabilistic

and if sleeping MTD selects zero rewards should be given. This can be achieved in above step 5 by selecting MTD i such that,

$$[P_i(t) \times \{\bar{r}_i(t) + \Delta_i(n)\}]_{max} = \left[P_i(t) \times \left\{\frac{R_i(t)}{n_i(t)} + \sqrt{\frac{s\log(t)}{n_i(t)}}\right\}\right]_{max}$$

$i \in k_t \subset M$

$P_i(t) = Probablity\ of\ MTD\ i\ to\ being\ active\ at\ time\ step\ t$
$i = MTD\ that\ has\ maximum\ value\ of\ P_i(t) \times \{\bar{r}_i(t) + \Delta_i(n)\}$
$k_t = set\ of\ MTD\ non\ empty\ queue\ buffer$
$M = set\ of\ MTD\ total\ MTDs$

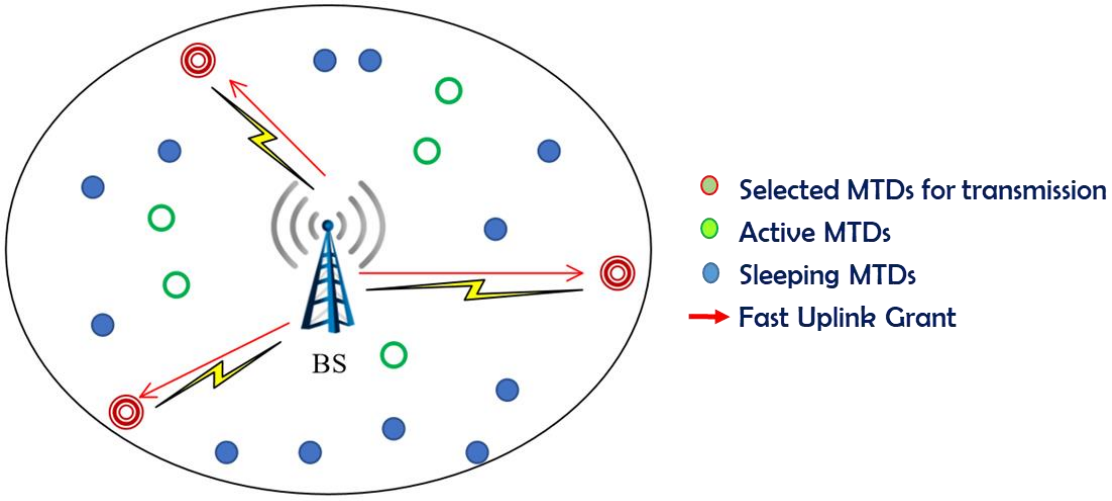


Figure 2.17 Sleeping MTDs and active MTDs

Because the proposed UCB algorithm gives allow to select MTDs with higher values of the utility function and a higher probability of being active while balancing the trade-off between exploitation and exploration. The proposed algorithm has the ability to provide system fairness by allowing the most important MTDs to be scheduled more often while other MTDs to be selected enough times to ensure the accuracy of the overall Fast Uplink Grant framework.

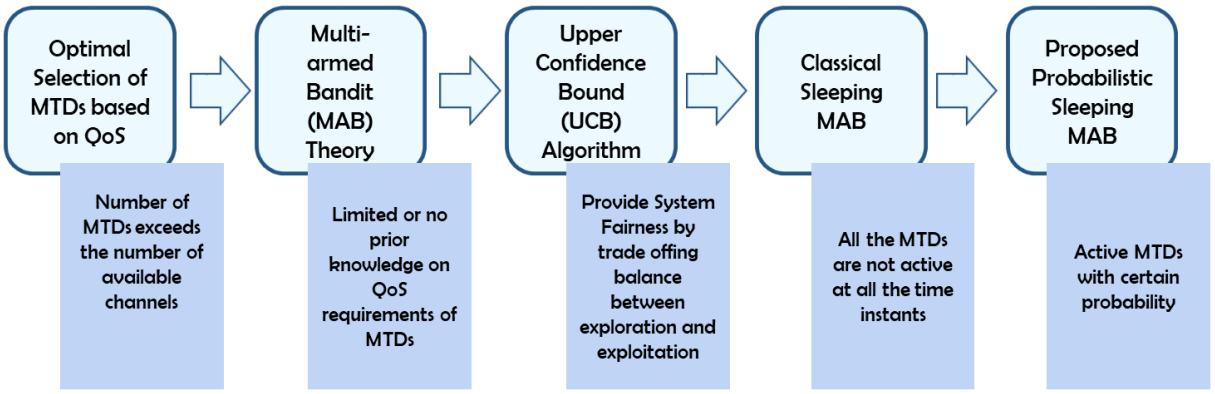


Figure 2.18 Summary of proposed optimal allocation framework

2.6 Machine Learning for Source Traffic Prediction

We did a small survey on the plausible machine learning algorithms that can be use source traffic predictions. When the transmission history of MTDs model as binary time series, a binary time series-forecasting algorithm can be used to enable source traffic prediction. There are numerous, machine learning models suitable for binary time-series predictions. However, as a control experiment, we surveyed a few machine-learning models that could be a possible choice for time series forecasting. A few of the available machine learning models are listed as follows.

- Hidden Markov Model (HMM)
- Artificial Neural Networks (ANN)
- Recurrent Neural Networks (RNN)
- Long Short Term Memory (LSTM)

2.6.1 Hidden Markov Model

An Hidden Markov Model (HMM) can be considered as a process where a doubly stochastic process with an underlying stochastic process that is not observable but can only be observed through another set of stochastic processes that produce the sequence of observed symbols. **[Rabiner]** HMMs are used to describe Markov chains of events. Markov chains are sequences of states where a given state only depends on the previous state but not what has happened before. However in an HMM, Markov chains of state sequences are hidden, hence the name Hidden Markov Models. All of the model behaviors can be shown as given in **[figure 2.19]**

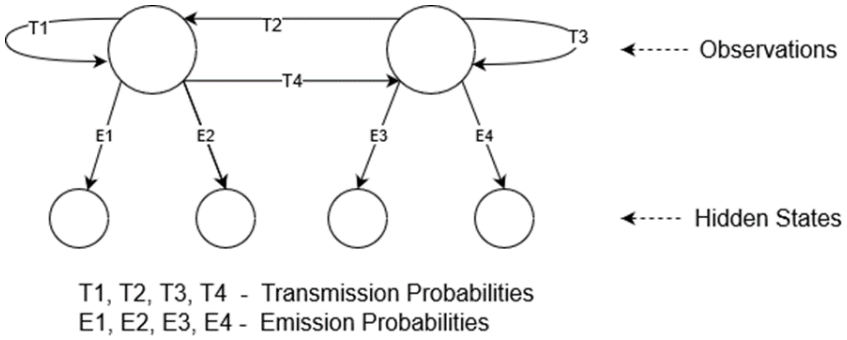


Figure 2.19 General architecture of a Hidden Markov Model

We had to do a control experiment using a model which only considers the periodic nature in order to emphasize that our LSTM model is actually considering the side information of event driven nature as a contributing factor to improve the accuracy. This is required as the data we used consists only of a Bernoulli's distribution where the transmissions are indicated. So the intention of this is to show what parts of the transmissions are predictable just considering the periodic nature of the dataset.

2.6.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are a sophisticated replication of the human brain which mimics the behavior of neurons to a certain extent. ANNs comprise nodes known as neuron as shown in **[figure 2.20]**. Researchers have successfully tried to tackle problems such as image recognition, natural language processing, stock price-trend forecasting, anomaly detection, recommender systems and many other implementations using artificial neural networks [Wang2003]. Thus ANNs are known as the universal function approximators.

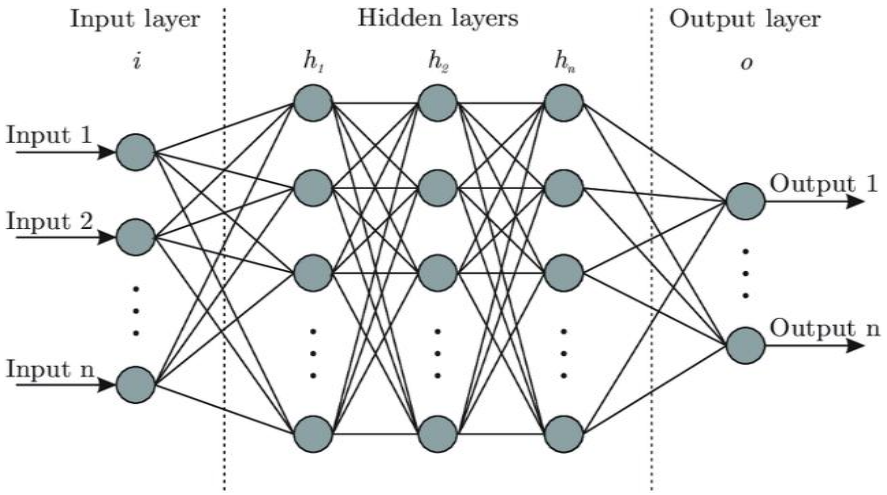


Figure 2.20 Structure of fully connected (vanilla) artificial neural network

However, the use of artificial intelligence and neural networks is relatively scarce in the telecommunication sector. This can be due to several reasons. First, artificial neural networks need a certain amount of pre-processing and training before applications. For that purpose, a rather large amount of computation power is necessary depending on the magnitude of the task. Base stations didn't possess such computation power in the past to address such an intensive training process. Also, there were other hardware-based methods to deliver better performance. However, with the development of technology, this hasn't been the case. With massive parallelization and high-performance computers, training models have become very efficient.

Even though the amount of computation power needed for training is large, after creating the model it can perform with relatively a lower amount of computations. This makes ANNs more appealing to be used in mission critical applications. Also it is beneficial for communication systems since the requirement for lower latencies has been their contemporary issue. Interpretability issues are another form of problems with ANNs that could arise in justifying a mission critical system. This is discussed in later chapters.

When considering deep learning algorithms as a modern update to ANNs that exploit abundant cheap computation. They are concerned with building much larger and more complex neural networks and, concerned with very large labeled datasets, such as image, text, audio, video and sequential data. **[Figure 2.21]** gives details about the most popular deep learning algorithms and their typical areas of application.

	Deep Learning Algorithm	Typical area of application
Supervised Learning	Artificial Neural Networks (ANNs)	Used for Regression and Classification
	Convolutional Neural Networks (CNNs)	Used for Computer Vision
	Recurrent Neural Networks (RNNs)	User for Time Series Analysis
Unsupervised Learning	Self-Organizing Maps (SOMs)	Used for Feature Detection
	Deep Boltzmann Machines	Used for Recommendation Systems
	AutoEncoders	Used for Recommendation Systems

Figure 2.21 Deep Learning algorithms and typical applications

The problem at hand is about predicting source traffic of MTC network. Since traffic has a sequential nature, a traditional dense architecture wouldn't be sufficient for a better prediction. In the recent

past the growth of RNNs; a specially designed neural network for time series data has proven to be very effective in the prediction scenarios. Therefore, we decided to try out the model using RNNs.

2.6.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) has become one of the most popular machine learning algorithms in the recent past. Unlike feedforward Neural Networks, whose neurons transmit information through the input layer, hidden layer and the connection of the output layer without any connection between the neurons in the same layer. But as shown in **[figure 2.22]** RNNs introduces the recurrent structure in the network and establishes the connection to the neuron itself. Because of this circular structure neurons can "remember" information from the previous moment (short term memory) in the neural network and influence the output of the current moment.

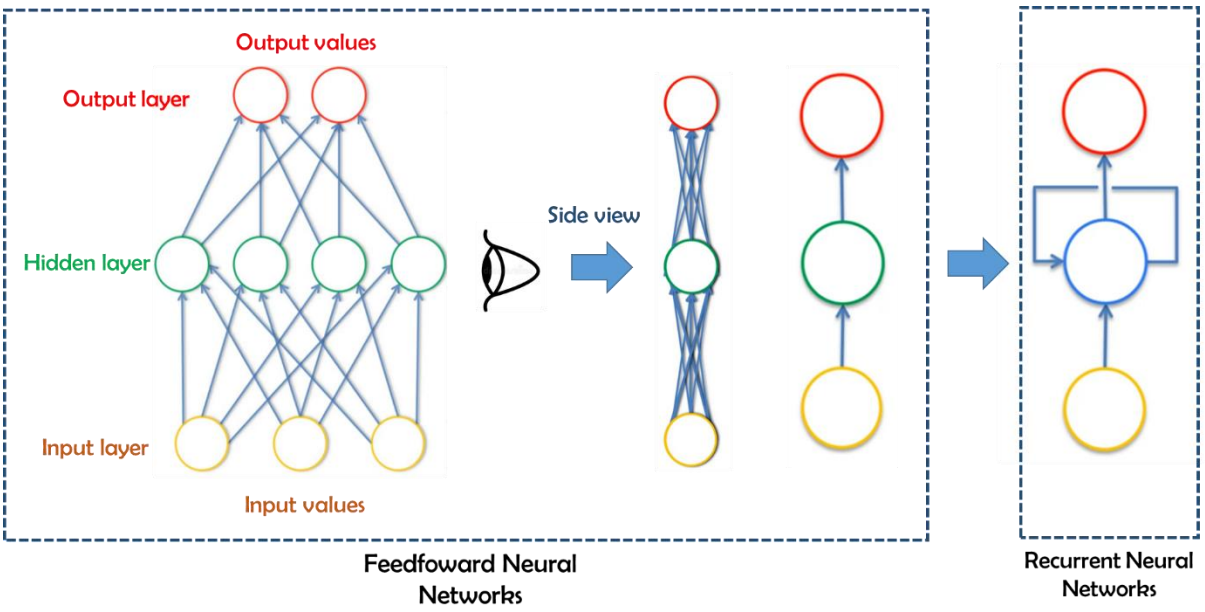


Figure 2.22 Recurrent Structure of RNNs

[Figure 2.23] graphically represent the unrolled recurrent structure of RNN in a time scale where neurons connect themselves through time. This gives the ability of RNNs to perform better on the data with time series **[Long Short-term Memory Neural Network for Network Traffic Prediction Qinzheng Zhuo, Qianmu Li*, Han Yan, and Yong Qi].**

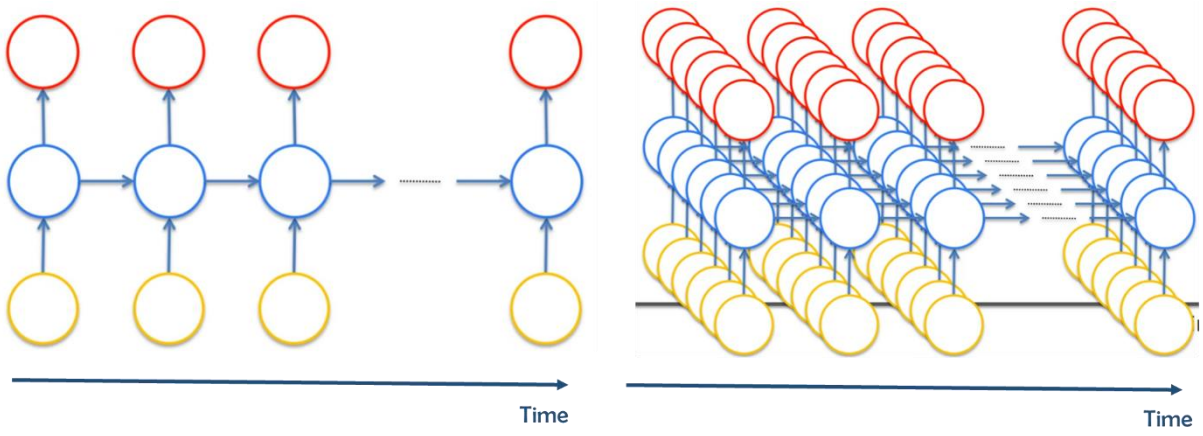


Figure 2.23 An unrolled recurrent neural network

In traditional RNNs as shown in **[figure 2.24]** the gradient signal can end up being multiplied by a large number of times by the $\mathbf{W}_{rec}$ weight matrix associated with the connections between the neurons of

the recurrent hidden layer. Hence the magnitude of the weights of the transition matrix can have a strong impact on the learning process. If the leading Eigenvalue of the weight matrix is smaller than one it leads to the vanishing gradient problem, where the gradient signal gets so small that learning slows down. If the leading Eigenvalue of the weight matrix is larger than one it leads to the exploding gradient problem which causes the learning to diverge **[Stock Transactions LSTMS SiyuanLiu].** This vanishing gradient problem in backpropagation graphically represents in **[figure 2.24]**. To solve this problem, HochReiter put forward a Long Short-Term Memory (LSTM) **[LSTM- Hochreiter].**

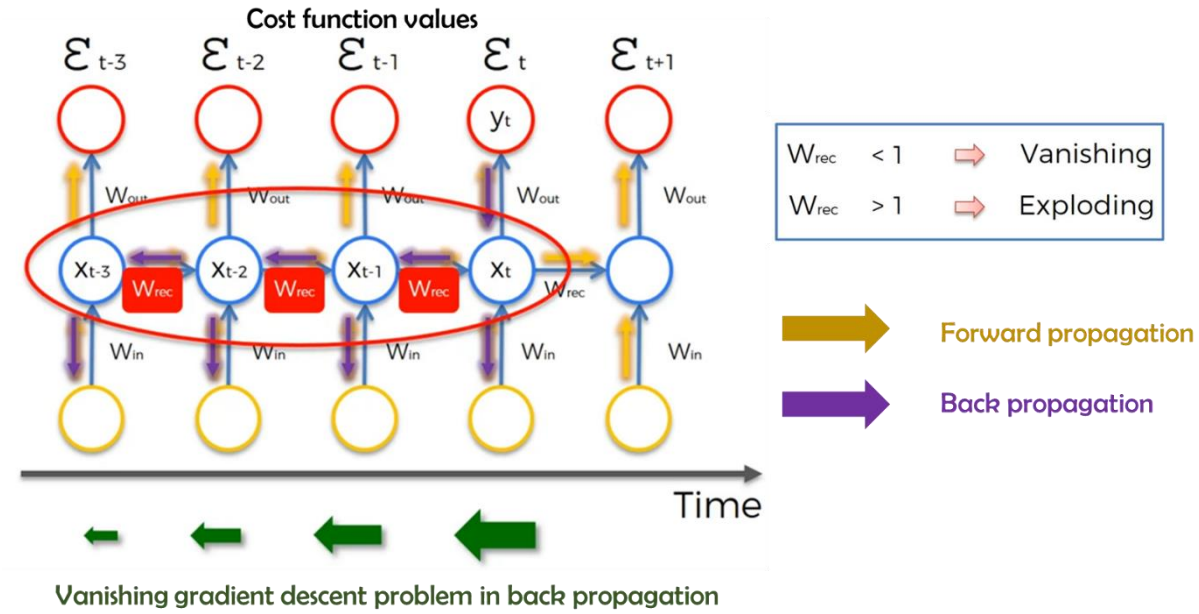
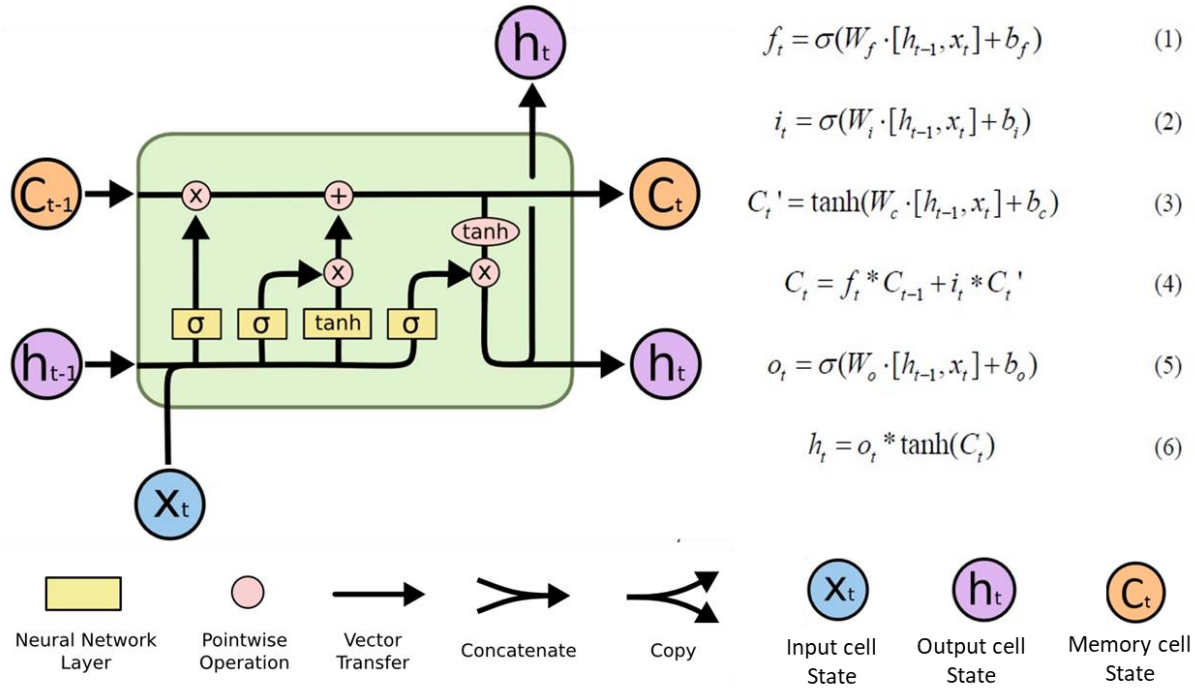


Figure 2.24 Vanishing gradient and exploding gradient problem associated with RNNs

2.6.4 Long Short Term Memory

Long short-term memory (LSTM) neural network is a variant of RNN. The key is to replace the neurons with cell states. Cell state is delivered over the timing chain, with only a few linear interactions, and information is easily maintained on cell units. As **[figure 2.25]** illustrates, each cell contains one or more memory cells and three nonlinear summation units. The nonlinear summation unit is also called the "gate", which is divided into 3 kinds: "Input gate" "Output gate" and "Forget gate". They control the input and output of memory cells by matrix multiplication. The forward propagation algorithm of LSTM is similar to RNN, and the input data is a time series of length T. **[Long Short-term Memory Neural Network for Network Traffic Prediction Qinzheng Zhuo, Qianmu Li*, Han Yan, and Yong Qi].**

RNNs are a modification to ANNs to solve problems where time-series data is involved. In other words, RNNs are specially designed to deal with temporal sequences of data **[Dorffner 96 neural networks].** However due to certain limitations such as gradient vanishing/exploding problems, more effective cells are used in the modern implementation known as LSTM (Long-Short Term Memory) cells. LSTMs have a high reputation in handling much longer sequences without coming across problems such as gradient explosion/vanishing. The typical structure of an LSTM cell is given in **[figure 2.25]**. The input cell state **(X_t)** represents the time series values as an array. There are two main outputs for an LSTM cell as opposed to one output in a general ANN cell. The memory cell state **(C_t)** is only transferred to the next cell while the hidden cell state **(h_t)** is transferred to through the layers. This architecture helps to keep the memory of longer sequences and generate outputs with dependence on time steps that occurred way back in time. Due to such characteristics exhibited in LSTM cells, we presumed that using this technology as the main focus for MTC source traffic could yield better results. The implementation details are given in chapter 3.



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (1)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (2)$$

$$C_t' = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (3)$$

$$C_t = f_t * C_{t-1} + i_t * C_t' \quad (4)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (5)$$

$$h_t = o_t * \tanh(C_t) \quad (6)$$

Figure 2.25 Structure of an LSTM cell

2.7 Hardware & Software Requirements of Machine Learning

According to the research methodology described in the previous chapter several machine learning and deep learning model implementations have done under source traffic predictions and optimal grant allocation parts of the project. Those implementations need a certain amount of pre-processing and training before getting more solid results for the analytical purposes of the project. Especially LSTM model data pre-processing, training and tuning tasks require a large amount of computation power and computing power available on general propose laptop computers was not sufficient for successful completion of these tasks. Therefore, we had to build, train and tune our machine learning models on could computing platform that provides scalable and affordable cloud computing service for machine learning and deep learning projects. As one of industry-recognized cloud computing services, we chose Amazon Web Service (AWS) Sagemaker as our cloud-computing platform. For the software and programing proposes of our machine learning and deep learning implementations, we decide to use Python programing language since it provides an extensive selection of machine learning-specific programming libraries.

2.7.1 Amazon SageMaker

Amazon SageMaker is a cloud machine-learning platform that enables data scientists and developers to create, train, and deploy machine-learning models in the cloud. It provides an integrated Jupyter authoring notebook instance for easy access to data sources for exploration and analysis, so user don't have to manage server side servers and configurations. Even through Sagemaker provide large variety of machine learning based cloud services during the project we only work with following Sagemaker features to develop our machine learning models.

Amazon SageMaker Notebooks instances is a fully managed machine learning compute instance running on the Jupyter Notebook App. Amazon SageMaker manages the creating of instance related computing and storage resources. This allow developers use Jupyter notebooks to prepare and process data, write code to train models, deploy and test or validate models.

Amazon SageMaker Studio is a web-based, integrated development environment (IDE) for machine learning that allow data scientists to build, train, debug, deploy, and monitor their machine learning models. Studio provides all the tools needed to take models from experimentation to production while boosting the productivity. In a single unified web interface, we can,

- Write and execute code in Jupyter notebooks
- Build and train machine learning models
- Deploy the models and monitor the performance of their predictions
- Track and debug the machine learning experiments

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the AWS cloud. Using Amazon EC2 eliminates investing in hardware up front, so we can develop and deploy applications faster. Amazon EC2 allows launching as many or as few virtual servers according to the requirement, configure security, networking, and manage storage. It also enables to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

Amazon Simple Storage Service (Amazon S3) is online storage service that design to make web-scale computing easier for developers. Amazon S3 has a simple web services interface that allow to store and retrieve any amount of data, at any time, from anywhere on the web. Before use Amazon automatic model training and tuning jobs, it is required to upload our dataset to S3 bucket.

Amazon Elastic Block Store (EBS) is an easy to use, high performance block storage service designed for use with Amazon Elastic Compute Cloud (EC2) for both throughput and transaction intensive

workloads at any scale. A broad range of workloads, such as relational and non-relational databases, enterprise applications, containerized applications, big data analytics engines, file systems, and media workflows are widely deployed on Amazon EBS. Even Machine Learning (ML) instance are tune off to minimize the cost, EBS volume keep stores the datasets and code of the project.

Amazon Elastic Inference (Amazon EI) is an accelerated compute service that allows attaching just the right amount of GPU-powered inference acceleration to AWS fully managed EC2 instances associate to the Jupyter notebooks. This allows choose the EC2 instance type that is best suited to the overall compute, memory, and storage needed to the application, and then separately configure the amount of inference acceleration that might require.

2.7.2 Machine Learning Instances

Amazon SageMaker provides a selection of instance types optimized to fit different machine learning (ML) use cases. As illustrated in **[figure 2.26],** instance types comprise varying combinations of CPU, GPU, memory, and networking capacity and allow flexibility to choose the appropriate mix of resources for building, training, and deploying your ML models. Each instance type includes one or more instance sizes, allowing you to scale your resources to the requirements of your target workload.

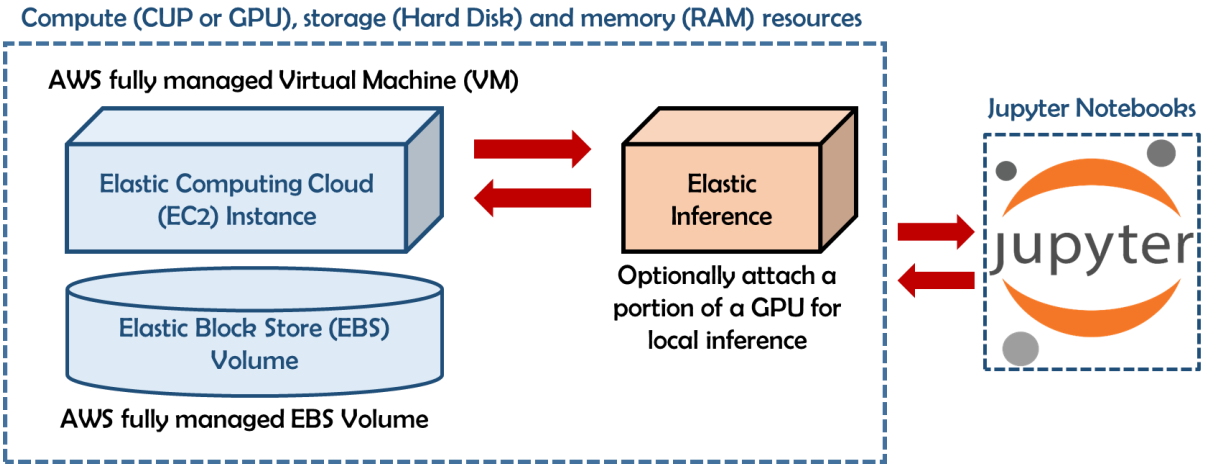


Figure 2.26 Machine Learning (ML) Instances

2.7.3 Sagemaker Model Training Jobs

Every model run on a Sagemaker training job has its own ephemeral computing cluster of single or multiple EC2 instances. That mean there is a dedicated EC2 instance alive for the number of seconds model needs to train. Where developer can build model on one EC2 instance and train on another EC2 instance. Especially, this ephemeral computing cluster automatically comes down immediately after model finished training by providing cost effective solutions. Following **[figure2.27]** shows Sagemaker-estimator configuration for training jobs where,

- **Algorithm container**
- **Execution role**
- **Number of dedicated EC2 instances for training job**
- **Type of EC2 instances for training job**
- **Size of EBS volume**

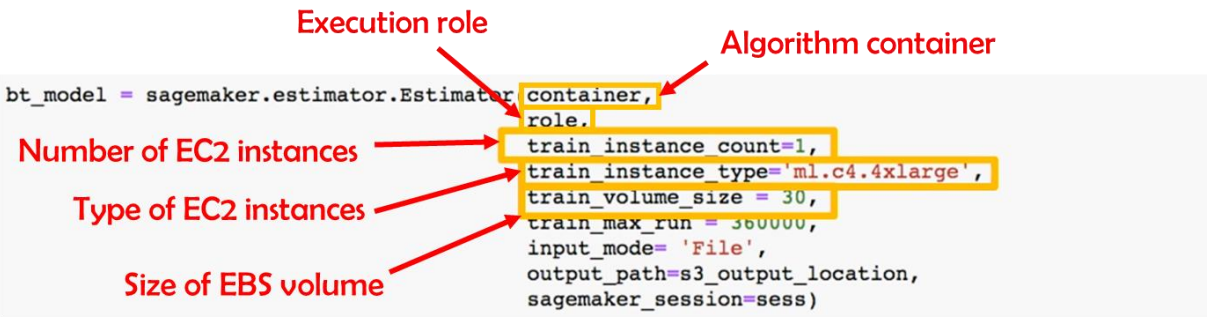


Figure 2.27 Sagemaker-estimator configurations

The following **[figure 2.28]** shows how to train machine learning model with Amazon SageMaker where area labeled Amazon SageMaker highlights the two components of Amazon SageMaker. To train a model in Amazon SageMaker, it is required to create a training job. The training job includes the following information,

- The URL of the S3 bucket where training data is stored.
- The compute resources that required for model training. These compute resources are ML compute instances that managed by Amazon SageMaker.
- The URL of the S3 bucket where output of the job need to be stored.
- The Amazon Elastic Container Registry path where the training code is stored.

After creating the training job, Amazon SageMaker launches the ML compute instances and uses the training code and the training dataset to train the model. It saves the resulting model artifacts and other output in the S3 bucket.

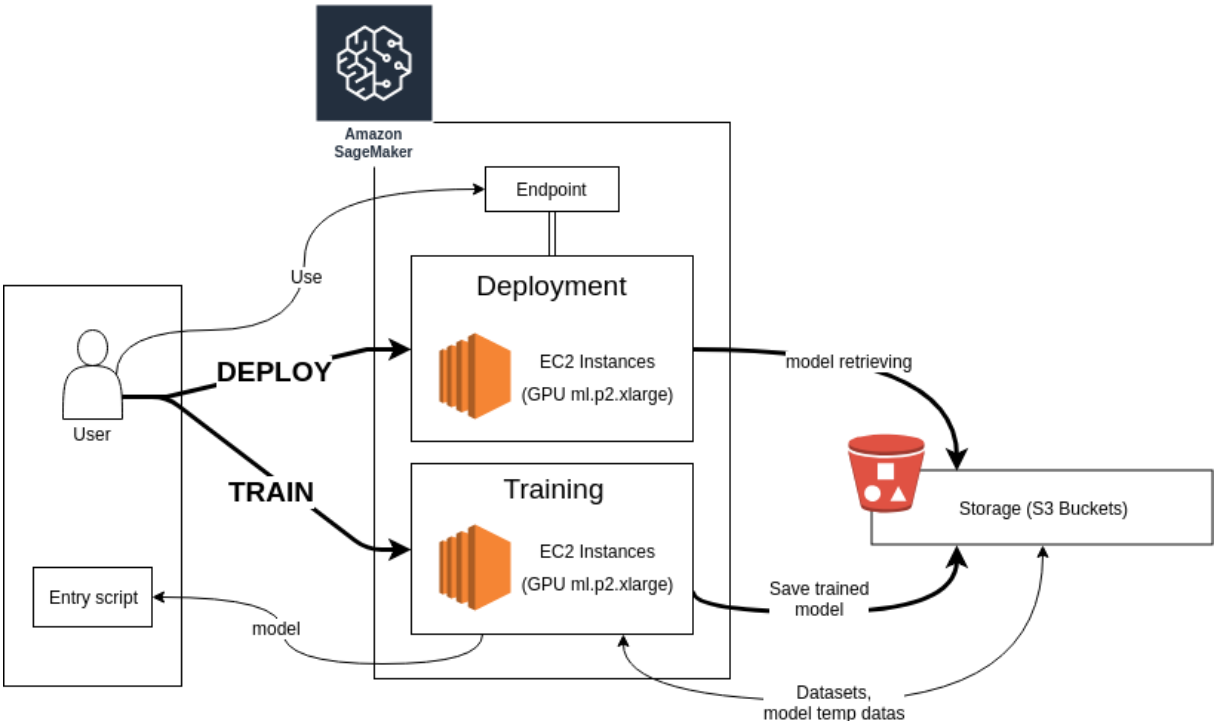


Figure 2.28 Train a Model with Amazon SageMaker

2.7.4 Hyperparameter Tuning

Generally, parameter is a model feature where hyperparameter is an extra term of real numbers that decides the structure of an artificial neural network. Deep learning models are composed of two different types of parameters.

- **Model parameters** are learned during the model training process
- **Hyper parameters** are parameters arbitrarily set before starting training process

The model parameters define how to use input data to get the desired output and they learned during the training time. Weights of the neural networks can be consider as typical example for model parameters. Inversely, hyperparameters determine how the model structured in the first place and they govern the entire training process. Learning rate, number of epochs, number of hidden units, number of hidden layers, activations function, loss function, and optimizer can be consider as typical examples for hyperparameter of an artificial neural network.

- **Learning rate** is the amount that the weights updated during training. Typically, learning rate has a small positive value, often in the range between 0.0 and 1.0.
- **Number of epochs** is the number of complete passes through the training dataset and it can be set to an integer value between one and infinity.
- **Batch size** is a number of samples processed before the model weights updated during the training process.
- **Activation functions** are mathematical equations that determine the output of an each neuron in the network where it determines whether neuron should be activated ("fired") or not, based on whether each neuron's input is relevant for the model's prediction.
- **Loss function**s are used to optimize the parameter values in a neural network model.
- **Optimizers** are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

Deep learning models tuning is a type of optimization problem where aim is to find the optimal set of hyper parameters and the right combination of their values which gives the minimum (eg. loss) or the maximum (eg. accuracy) value of an objective function. **[Figure 2.29]** shows the different between regular model training process and hyperparameter tuning process.

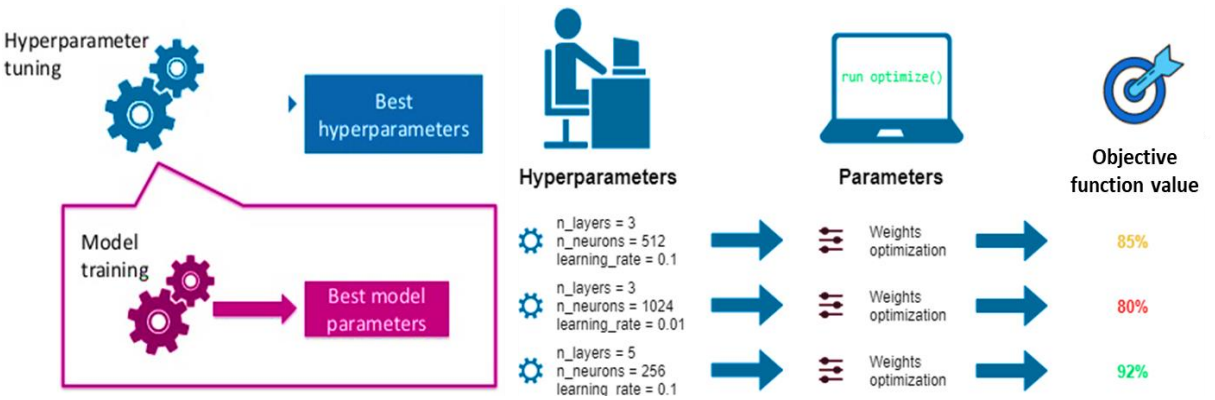


Figure 2.29 Hyperparameter tuning vs. Model training

Even though varies hyperparameter tuning methods exists

- Manual Search
- Grid Search
- Random Search
- Bayesian Optimization

When using Manual Search, developer can choose some model hyperparameters based on their judgment/experience. Then train the model, evaluate its accuracy and start the process again. This loop is repeated until a satisfactory accuracy is scored.

In Grid Search, we set up a grid of hyperparameters and train/test our model on each of the possible combinations. In order to choose the parameters to use in Grid Search, we can now look at which parameters worked best with Random Search and form a grid based on them to see if we can find a better combination. Grid Search can be implemented in Python using scikit-learn GridSearchCV() function. Also on this occasion, I decided to divide our training set into 4 Folds (cv = 4). When using Grid Search, all the possible combinations of the parameters in the grid are tried. In this case, 128000 combinations (2 × 10 × 4 × 4 × 4 × 10) will be used during training. Instead, in the Grid Search example before, just 80 combinations have been used.

In a random search, hyperparameter tuning chooses a random combination of values from within the ranges that you specify for hyperparameters for each training job it launches. Because the choice of hyperparameter values doesn't depend on the results of previous training jobs, you can run the maximum number of concurrent training jobs without affecting the performance of the search.

Grid Search is slower compared to Random Search but it can be overall more effective because it can go through the whole search space. Instead, Random Search can be faster fast but might miss some important points in the search space.

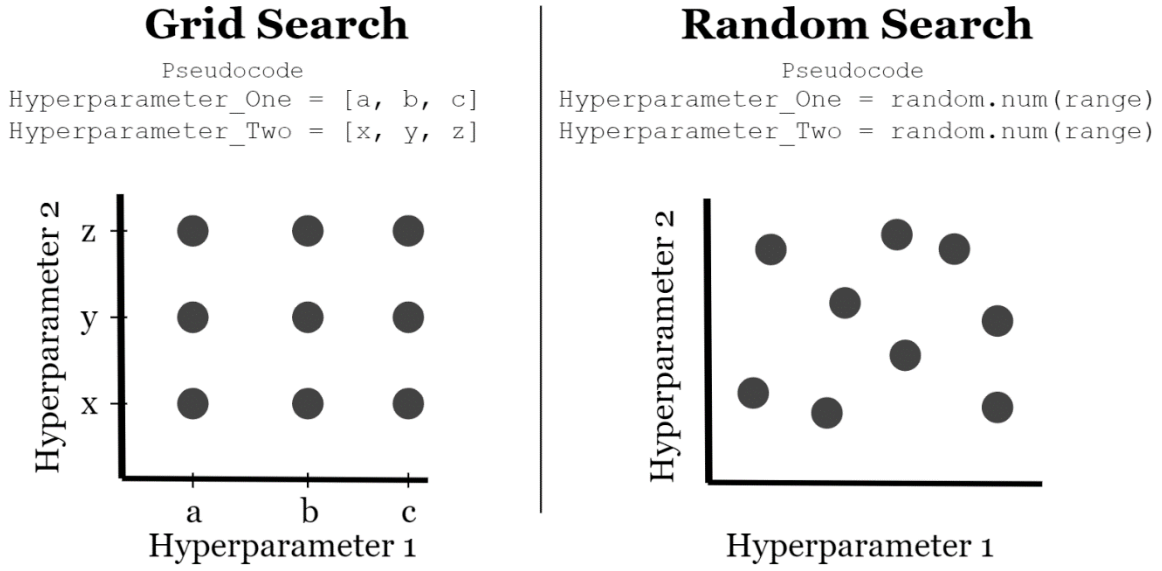


Figure 2.30 Grid Search vs. Random Search

2.7.5 Bayesian Optimization

Bayesian search treats hyperparameter tuning like a regression problem. Given a set of input features (the hyperparameters), hyperparameter tuning optimizes a model for the metric that you choose. To solve a regression problem, hyperparameter tuning makes guesses about which hyperparameter combinations are likely to get the best results, and runs training jobs to test these values. After testing the first set of hyperparameter values, hyperparameter tuning uses regression to choose the next set of hyperparameter values to test.

Bayesian Optimization can be performed in Python using the Hyperopt library. Bayesian optimization uses probability to find the minimum of a function. The final aim is to find the input value to a function which can give us the lowest possible output value.

Bayesian optimization has been proved to be more efficient than random, grid or manual search. Bayesian Optimization can, therefore, lead to better performance in the testing phase and reduced optimization time.

In Hyperopt, Bayesian Optimization can be implemented giving 3 three main parameters to the function fmin().

- Objective Function = defines the loss function to minimize.
- Domain Space = defines the range of input values to test (in Bayesian Optimization this space creates a probability distribution for each of the used Hyperparameters).
- Optimization Algorithm = defines the search algorithm to use to select the best input values to use in each new iteration.

Additionally, can also be defined in fmin() the maximum number of evaluations to perform.

Bayesian Optimization can reduce the number of search iterations by choosing the input values bearing in mind the past outcomes. In this way, we can concentrate our search from the beginning on values which are closer to our desired output.

2.7.6 Sagemaker Model Tuning Jobs

For Hyperparameter tuning we use an Amazon SageMaker implementation of Bayesian optimization. When choosing the best hyperparameters for the next training job, hyperparameter tuning considers everything that it knows about this problem so far. Sometimes it chooses a combination of hyperparameter values close to the combination that resulted in the best previous training job to incrementally improve performance. This allows hyperparameter tuning to exploit the best known results. Other times, it chooses a set of hyperparameter values far removed from those it has tried. This allows it to explore the range of hyperparameter values to try to find new areas that are not well understood. The explore/exploit trade-off is common in many machine learning problems.

To specify settings for the hyperparameter tuning job, we define a JSON object. Then pass the object as the value of the HyperParameterTuningJobConfig parameter to CreateHyperParameterTuningJob when you create the tuning job.

In this JSON object, we specify:

- The ranges of hyperparameters that you want to tune. For more information, see Define Hyperparameter Ranges
- The limits of the resource that the hyperparameter tuning job can consume.
- The objective metric for the hyperparameter tuning job. An objective metric is the metric that the hyperparameter tuning job uses to evaluate the training job that it launches.

2.7.7 Python Programming Libraries for Machine Learning

For the programing and coding proposes of our machine learning and deep learning implementations, we decide to use Python programing language since it provides an extensive selection of machine learning-specific programming libraries. Python machine learning libraries have grown to become the most preferred language for machine learning algorithm implementations. **[Figure 2.31]** shows main Python libraries used for machine learning and deep learning purposes.

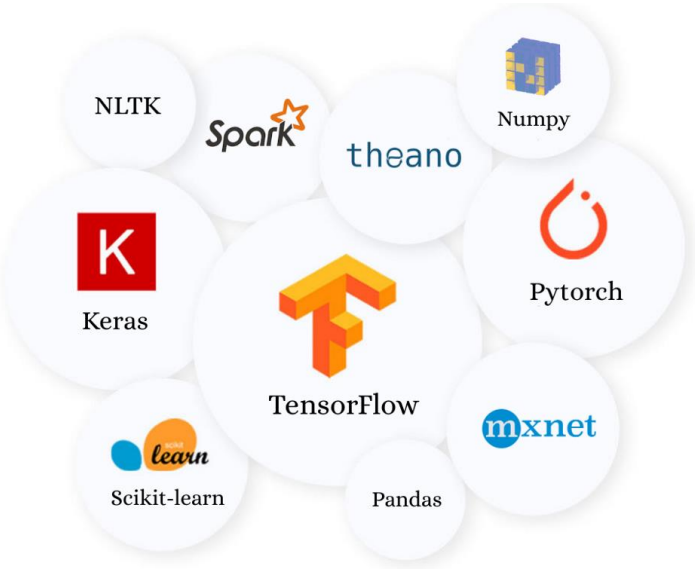


Figure 2.31 Python Machine Learning Libraries

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for groping, combining and filtering data.

Matpoltlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar chats, etc,

Skikit-learn is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or Theano. It can run seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network. One of the best thing about Keras is that it allows for easy and fast prototyping.

3. PROPOSED FAST UPLINK GRANT FRAMEWORK

Consider a cellular-enabled Machine Type Communication (MTC) network composed of a Base Station (BS) and M number of MTDs where MTDs are stationary or exhibits low mobility. If the MTDs have been transmitting through the traditional RA method. We assume that the BS has gathered MTD transmission history over time by keeping records about RA scheduling requests and then historical transmission data model as a binary time series.

In the case of periodic traffic prediction, this historical transmission data can be used to learn time instants during which the MTDs have data to transmit using the proposed source traffic prediction mechanism. After completely learning the periodic transmission patterns, RA scheduling requests will no longer be needed **[DI paper],** for allocating the uplink resources to periodical transmissions. However, optimally selecting which MTDs should be allocated has to be done using an intelligent method such as Sleeping Multi-Arm Bandit Theory **[MAB paper].**

On the other hand, in the case of event-driven traffic **[Learning how to communicate in the internet of things: Finite resources and heterogeneity]**, the concept of fast uplink grant proposes not to completely abandon RA scheduling requests even after the prediction patterns are fully learned by the BS. Hence any scheduling request is seen as an event trigger by the BS. When an IoT event occurs, some MTDs will initiate access before others. This is a practical assumption since most IoT events propagate through geographical areas and the sensors capture them at different times **[Learning how to communicate in the internet of things: Finite resources and heterogeneity]**. Consider a K number of MTDs that will transmit new data packets to detect an unexpected event. Such an event that is not seen before is impossible to predict by the base station just by looking at the transmission history. Therefore, RA method can be used to report unusual and random traffic generated by MTDs with high entropy.

The ultimate objective of this project is to introduce an innovative machine-learning based framework that can contribute to the realization of Fast Uplink Grant. **[Figure 3.1]** represents the proposed Fast Uplink Grant framework as a conceptual block diagram where MTDs and the BS actively participate in the process. From the Machine Type Device (MTD) side, whenever MTD has data packets to transmit it should undergo the following procedure to receive the Fast Uplink Grant,

1. As a very first step, MTD should enter into active mode from its sleeping or ideal mode where it listens to the broadcast channel to retrieve the Fast Uplink Grant allocation information.
2. After that, if data packets generate according to predefined periodic intervals that data packets can transmit after Fast Uplink Grant received. The source-traffic prediction framework already has learned periodic time intervals, which MTDs have data to transmit.
3. If data packets generate to report an ongoing MTC event, MTD needs to wait a predefined time interval until BS allocates a Fast Uplink Grant without immediately sending an RA request.
4. If MTD received Fast Uplink Grant within the waiting time, MTD can transmit data packets using the Fast Uplink Grant (traffic channel) allocated by BS
5. If Fast Uplink Grant did not receive within the waiting time, MTD needs to send an RA request to the BS by following a traditional way.
6. After sending RA request MTD required to wait until BS allocates Fast Uplink Grant.

The Base Station (BS) side is responsible for following tasks,

1. BS should be able to perform source traffic prediction based on previously seen event-driven and periodic traffic patterns.
2. BS should be able to select the optimal set of MTDs for Fast Uplink Grant allocation based on their latency requirements.

3. When BS received RA requests due to a new event. BS needs to train its source-traffic prediction framework to identify other MTDs that experience the same event and their transmission order. Furthermore, new event reporting MTDs need to allocate Fast Uplink Grant considering their latency requirements.
4. When BS received RA requests due to previously observed events. BS needs to predict the other MTDs that experience the same event and their transmission order. After that optimal Fast Uplink Grant allocation will be done based on latency requirements.
5. After optimal Fast Uplink Grant allocations performed BS needs to broadcast that channel allocation information using broadcast channels.

As illustrated in **[figure 3.1]** proposed Fast Uplink grant framework has facilitated the connection between BS and MTD using following wireless logical channels and their functions,

- Random Access Channel (RCCH) – send random access requests to the base station
- Traffic Channel (TCH) - transmit MTD data packet to the base station
- Broadcast Channel (BCCH) – Broadcast Fast Uplink Grant channel allocation information to MTC network

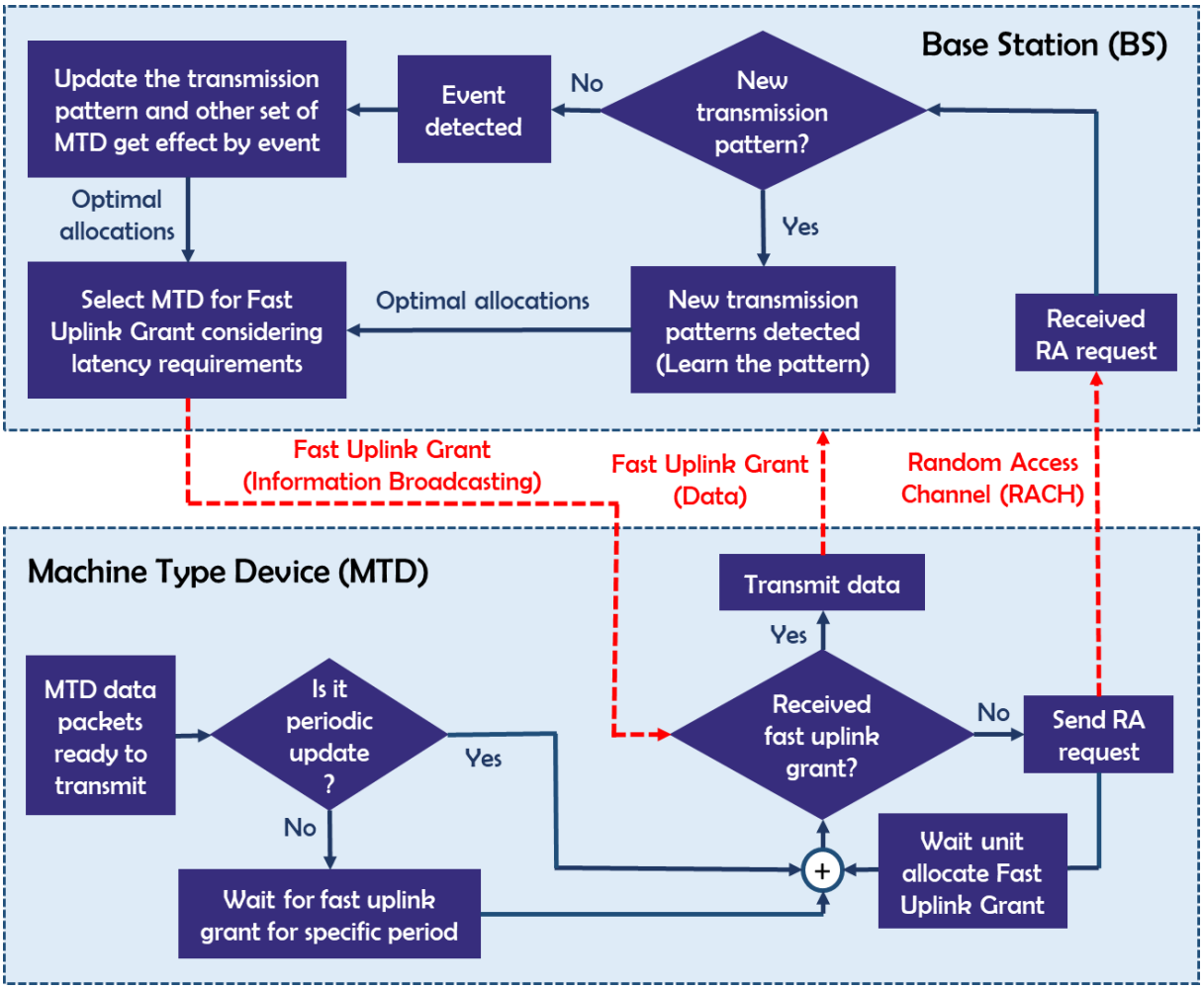


Figure 3.1 Proposed Fast Uplink Grant Framework

3.1 Source Traffic Prediction using LSTM Networks

Recurrent Neural Networks (RNNs) and LSTMs are widely used as an effective model for identifying patterns and forecasting sequential applications. LSTMs have overcome vanishing gradient and exploding gradient issues of traditional RNNs using a memory cell, which can maintain its state over time, and nonlinear gating units, which regulate the information flow into and out of the cell. After considering all those advantages, LSTMs were selected as the main model for source traffic prediction.

According to **[figure 3.2],** as the first step of implementing of LSTM based source-traffic prediction framework, the MTC network simulation program was built to generate artificial MTD transmission data. This MTC network-simulating program can generate MTD transmission data as a binary time series. Therefore, this generated transmission data considered as the MTD transmission history and use to train, validate, tune and test the LSTM model. Then initial LSTM model was build using Tensorflow blackened Keras python library **[Keras documentation]**. After that data pre-processing was done to improve the quality of the raw transmission data before data fetch to the LSTM model. LSTM model validation was an important step of this implantation since it verifies the LSTM model does not overfit on the training data. Finally, model tuning was done to find the optimal set of LSTM hyperparameters that gives the maximum validation accuracy score. Overall LSTM implementation can be summarized using the block diagram shown in **[figure 3.2].**

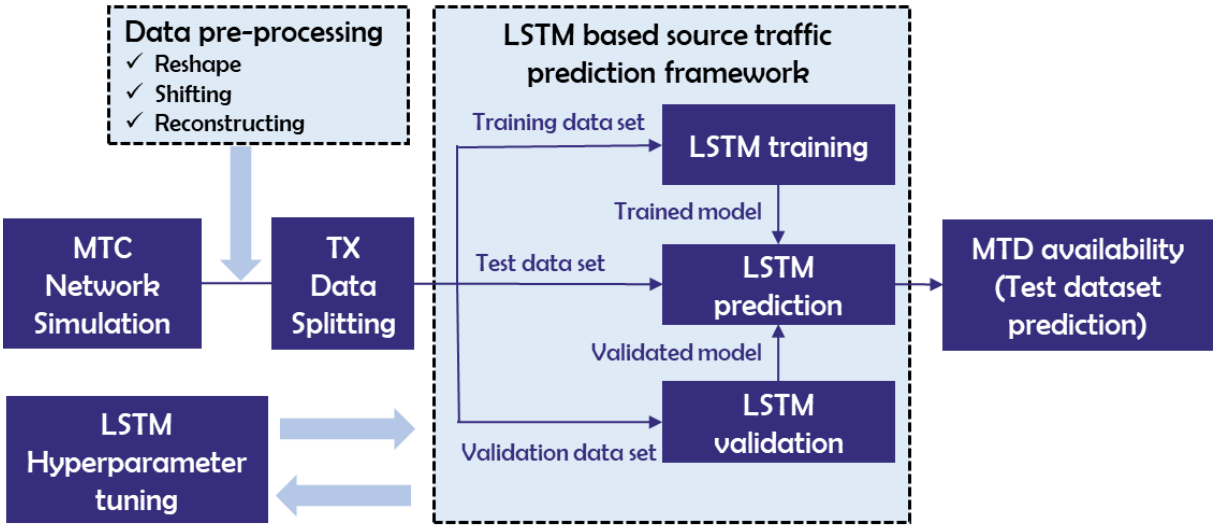


Figure 3.2 Implementation of LSTM based source-traffic prediction framework

3.1.1 MTC Network Simulation

Simple Python program was written to generate MTD transmission data by modeling the transmission history of each MTD as a binary time sequence where packet transmission is presented by 1 and being silent with 0 **[DI paper]**. This program uses to generate periodic, event-driven and random MTD transmission patterns. NumPy library uses to write this program because it supports multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays **[NumPy documentation]**. To keep the system simple for analytical purposes, small MTC network with five MTDs were considered (X, Y, Z, T, and W) for the data generation. (Later on, MTC network simulation was extended to MTC network with 100 MTDs) The length of the sequence of an IoT event is assumed to be 12 timesteps.

For MTD X, transmission happens at time steps $t \in \{3, 4, 5, 6, 9, 10\}$. In other words, at time steps 3, 4, 5, 6, 9 and 10, the MTD may or may not transmit. The probability of X to transmit at any of the given time steps is assumed to be constant in this case. Nevertheless, during the remaining time steps, it's

assured that the MTD will be in a state of sleep where no transmissions occur. An example transmission pattern of MTD X can be X12 = {001101001000}.

Python code used to generate the MTD X transmission time series (Xtrans) included under Appendix-A **[section 2.1]**-MTC X (periodic transmission). The **random.choices(population, weight, k)** inbuilt function of the random module use to get periodic-binary series where population = [1, 0], weights = [P,1-P], and k = length of the time series. (P = probability of transmissions, 1-P = probability of being silent). Then transpose and reshape functions of NumPy library use to get the exact (12000, 1) shape to MTD X time series.

For MTD Y, the random transmission happens at any time. In other words, there can be a transmission at any given timestep.

Python code used to generate the MTD Y transmission time series (Ytrans) included under Appendix-A **[section 2.2] -** MTC Y (random transmission). The **numpy.random.randint(low, high)** inbuilt function of the NumPy library use to get random number series within the (low, high) interval. To obtain MTD Y transmissions as binary random time series we use low = 0 and high = 1. Finally, np.reshape function uses to reshape Ytrans from (12000,) to (12000, 1) shape.

Y causes T and W to transmit directly and indirectly respectively. All together, Z, T, and W exhibit an event-driven transmission nature correlated to X and Y in the specific order. As mentioned in Fig. 3, for Z, we assume that the transmission occurs 3 timesteps after X with a probability of 0.7. For MTD T, the transmission happens 2 timesteps after X with a probability of 0.7 as given in Fig. 4. Finally, for W, we assume that the transmission occurs one timestep after T with a probability of one. By adopting such a structure we can cover three levels of entropy; completely random (Y), partially uncertain (Z and T), and completely certain (W). Also in W, we can see a 2nd-degree causality with Y.

Python code use to generate MTD Z, T and W transmission time series has include under Appendix-A **[section 2.3] -** MTD Z, T and W (event-driven transmission). The **random.choices(population, weight)** inbuilt function and if statements use to get binary series correlated to MTD X and MTD Y time series where population = [1, 0] and weights = [P,1-P] (P = correlation probability).

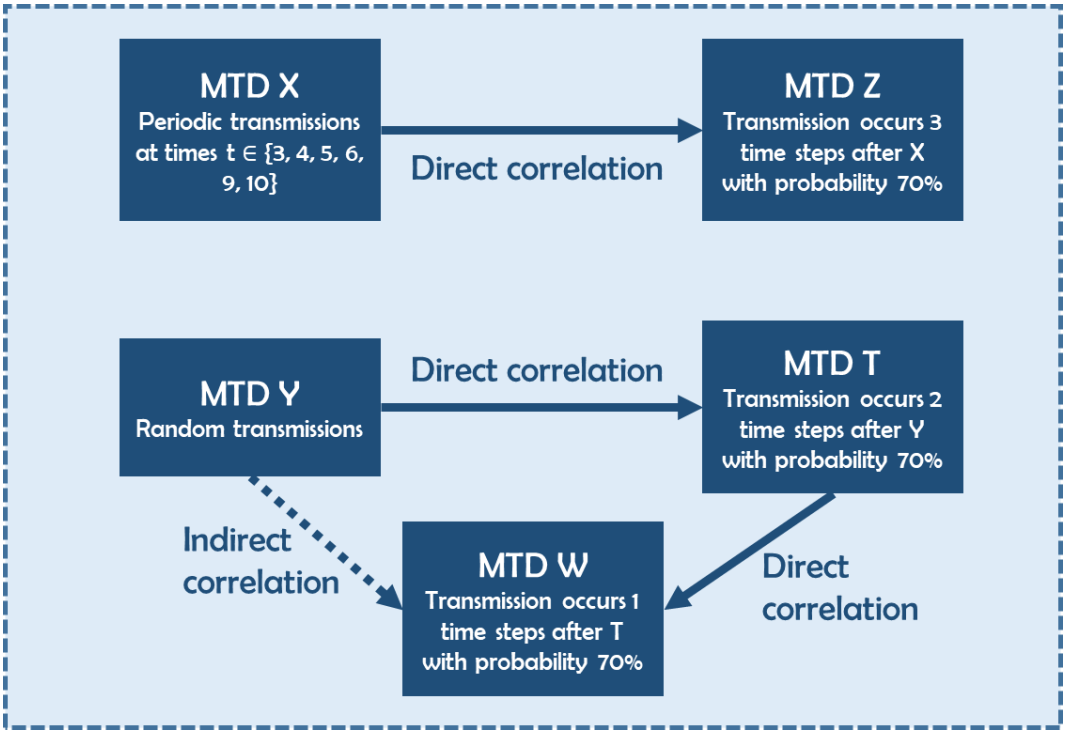


Figure 3.3 MTC Network Simulation

Finally, all 5-MTD transmission time series concatenate into MTD-Tx array with (12000, 5) and save them into MTD_Tx.csv file.

3.1.2 Data Preprocessing

Before feeding data to the LSTM, the generated data is reshaped most optimally for the model to understand. Generally, the input of an LSTM takes the shape of a 3D array. The X, Y, and Z-axis of the array represent the length of the sequence in time steps, the number of sequences and the MTDs are respectively as illustrated in **[figure 3.4]**. All the event sequences from an MTD are reshaped into a vector and then they are arranged in a 2D matrix where the width is decided by the number of time steps input to the LSTM. This value is also fine tuned under the hyperparameter tuning process. As an example structure, **[figure 3.4]** shows a modified sequence length of 12.

The next single time step after the first 12, is taken as the labels in the generated transmission data. They are given as 13th, 14th, 15th, etc in **[figure 3.4]** for MTD X. During the training process, the transmission pattern of the previous 12 timesteps is fed to the LSTM as the independent feature matrix alongside with the labels. It was then tuned later on to yield the best performing model.

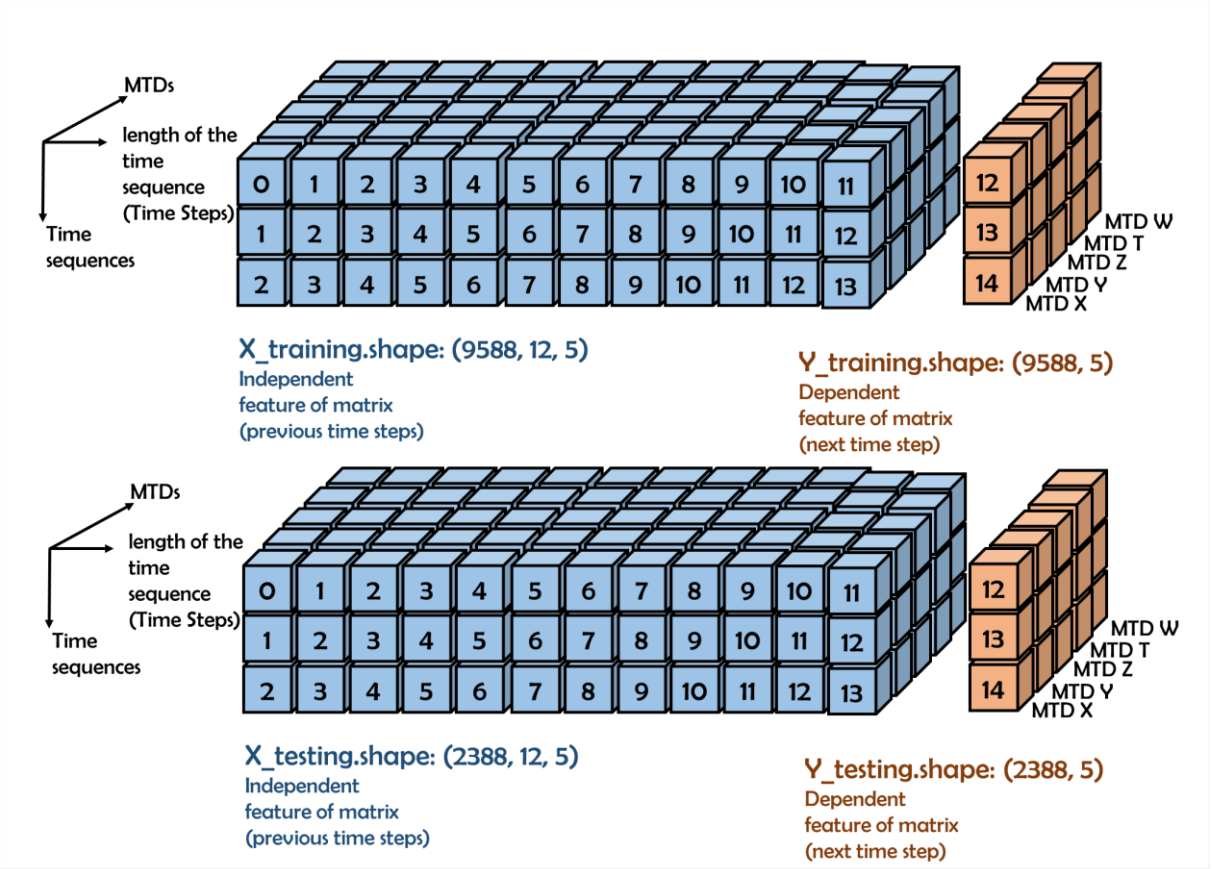


Figure 3.4 LSTM input data Structure

Python codes use for data preprocessing included in Appendix-B**.** First transmission data split into training and testing datasets where 80% of transmission data use for LSTM model training and rest 20% use for LSTM model testing purposes. Then NumPy inbuilt array manipulation functions such as reshape, append, delete and indexing apply on (9600, 5) shape training dataset to get independent and dependent training-feature of the matrix with the shape of (9588, 12, 5) and (9588, 5) respectively. Similar way the same set of functions apply on (2400, 5) shape testing dataset to get independent and depended testing-feature of the matrix with the shape of (2388, 12, 5) and (2388, 5) respectively. Finally, all 4 matrix save as NPZ files by **np.savez** function.

3.1.3 LSTM Model Building and Training

The LSTM model builds using Tensorflow blackened Keras library. The first LSTM layer outputs hidden states for each input time step and this continues until the final LSTM layer. However, the passing of hidden states is done through a dropout regularization layer where 20% of the hidden states are dropped before sending the values. In the final LSTM layer, after dropout regularization, it only outputs one hidden state per sequence. Finally, a dense layer is used to compare labels with predictions.

Even though there are many metrics for loss functions used in neural networks, we selected the mean square error (MSE) function for this application. Usually, MSE is the preferred choice for LSTMs as it explicitly gets the deviation of the prediction from real values. But in this case, MSE was selected as we are more interested in a better classification for future timesteps to be transmitting or not transmitting (sleeping). So, for the matter at focus, MSE was used as it gives a sense of direction from absolute random (0.5 probability) to either side of the output probability. Also, since MSE takes the squared difference, it gives high weights to large errors. Therefore, the weights can be penalized properly to drive towards the correct direction of the prediction. The equation of MSE is given by Equation 1.

$$MSE = \frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}$$

Where N is the number of training samples.

For adaptive learning rate optimization, we use Adam optimizer algorithm **[Adam: A method for stochastic optimization]** since it can leverage the power of adaptive learning rate methods by finding individual learning rates for each parameter. We use binary accuracy as the model training metric since our interest is in predicting a Bernoulli's distribution with our regression model.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 12, 26)	3328
dropout_1 (Dropout)	(None, 12, 26)	0
lstm_2 (LSTM)	(None, 12, 15)	2520
dropout_2 (Dropout)	(None, 12, 15)	0
lstm_3 (LSTM)	(None, 15)	1860
dropout_3 (Dropout)	(None, 15)	0
dense_1 (Dense)	(None, 5)	80

Figure 3.5 Initial LSTM Model Structure

Appendix-C contains Python codes related to LSTM model building and training tasks. Training of the model was done in two main ways. The model was first trained with arbitrary parameters and then it was trained using a Bayesian Optimizer. Initially build LSTM model structure shows in **[figure 3.5]** and the layer sizes were obtained based on the literature **[Learning capability and storage capacity of two-hidden layer feedforward networks]** where the first layer and hidden layer cell numbers were taken using an equation. The rest of the parameters are added arbitrarily or according to the general convention. This model was just used as a performance comparison to identify the difference between

any performance improvement obtained with hyperparameter tuning. **[figure 3.6]** and **[figure 3.7]** visualize the accuracy and loss variation of the very first training process.

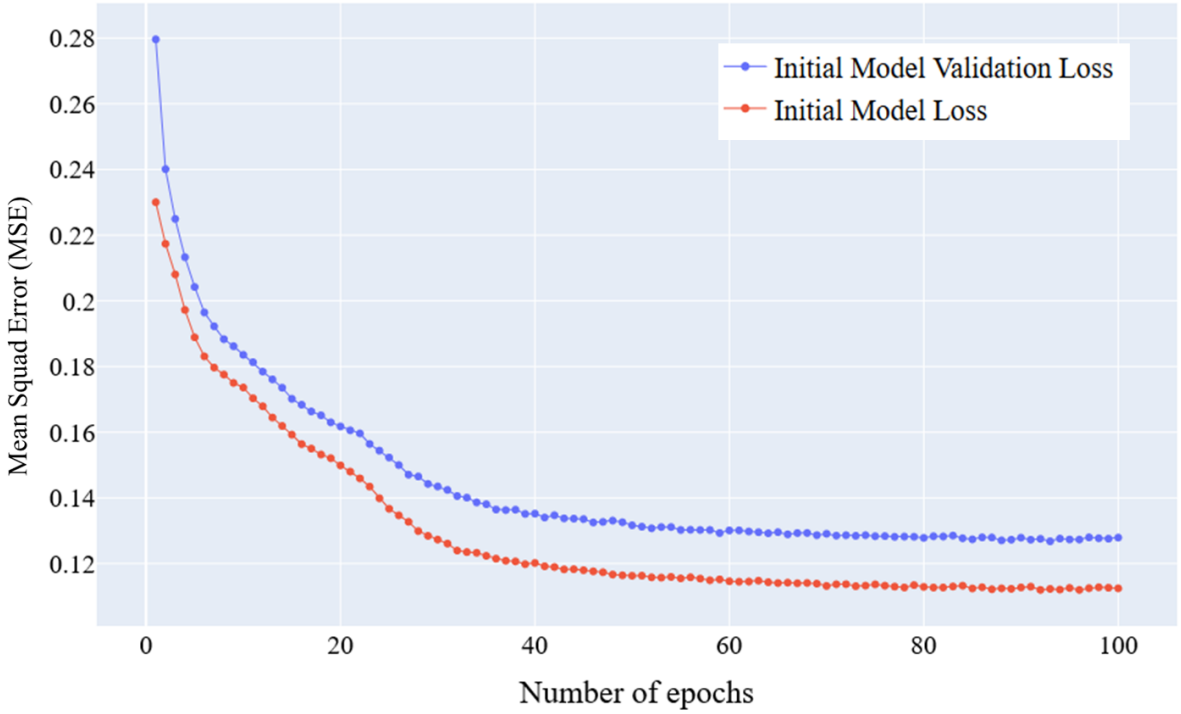


Figure 3.6 Initial model MSE and validation MSE variation during training process

According to **[figure 3.6]** Mean Squad Error (MSE), almost get converged after 80th epoch and mean squad error calculated based on training dataset is much lower than mean squad error calculated on the testing dataset. Similarly, **[figure 3.7]** shows binary accuracy variation during the initial model training process and it is converged after the 80th epoch.

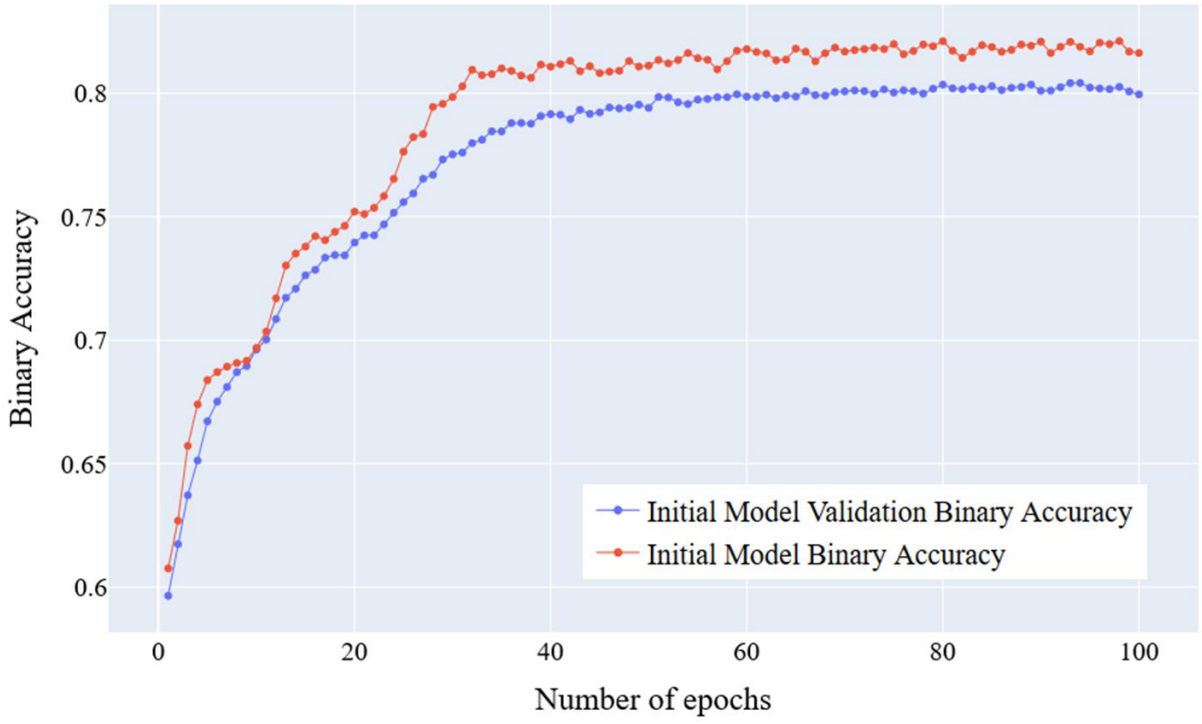


Figure 3.7 Initial model binary accuracy and validation binary accuracy variation during training process

3.1.4 LSTM Model Tuning

The bayesian optimizer is used for hyper-parameter tuning and validation purposes. It has a high reputation on finding the optimal set of hyper-parameters for any model from a given range of values more efficiently **[19th reference of our paper]** than other traditional methods such as manual method, grid search, and random search methods. Internally, Bayesian hyper-parameter optimizer builds a probability model of the objective function and uses it to select the most promising hyper-parameters to evaluate the true objective function. By setting binary accuracy as the objective function, we have given pragmatic ranges for hyper-parameters of the model.

Model tuning was done with two main intentions. First, to fine tune the model by determining optimal LSTM cell numbers for input layer and hidden layer to obtain the best validation binary accuracy. Secondly, to obtain the optimal length of restructured sequence length.

Bayesian optimizer within Keras hyperparameter tuner library used for very first model tuning job where optimal LSTM model structure determined. Tuners are in Keras tuner library used to do the hyperparameter search. It is possible to create custom Tuners by subclassing **kerastuner.engine.tuner.Tuner. [Appendix-D]** gives Python codes related to this tuning job. After importing basic data processing libraries and installing Keras tuner library, pre-processed MTD transmission data load using NPZ files. Then CSVLogger function has been used for saving the training log of each tuning trial as a CSV file for later tuning results analysis.

Table 3.1 Scored binary accuracy (objective function) and the LSTM cell structure of tuning trials

Tuner trial number	Score	Input layer units	Number of hidden layers	Hidden layer 1 number of units	Hidden layer 2 number of units	Hidden layer 3 number of units
0	0.832580	40	2	10	200	0
1	0.830988	70	2	190	50	0
2	0.832663	190	3	130	10	40
3	0.833333	20	1	200	0	0
4	0.833668	150	1	100	0	0
5	0.832580	10	3	60	140	20
6	0.832580	190	1	10	0	0
7	0.832663	190	3	70	200	140
8	0.832328	30	1	20	0	0
9	0.833668	190	2	180	200	0
10	0.834506	10	1	170	0	0
11	0.831575	200	3	130	10	180
12	0.832328	20	3	200	160	190
13	0.834673	150	1	170	0	0
14	0.831910	160	3	10	170	40
15	0.833417	130	1	80	0	0
16	0.831742	180	3	20	40	190
17	0.833417	10	1	70	0	0
18	0.830737	50	3	190	30	200
19	0.831742	50	1	20	0	0

After that **[Appendix-E]** shows codes related to Hyper Parameter Optimization (HPO) result analysis. **[Table 3.1]** shows the scored binary accuracy (objective function) of the first 20 tuning trials including the LSTM cell structure selected by Bayesian optimizer. Details about all the tuner trials can find under **[Appendix-E]** section 1.3. **[figure 3.8]** shows the variation of maximum validation binary-accuracy for each tuning trial where the maximum number of tuning trials was set to 50.

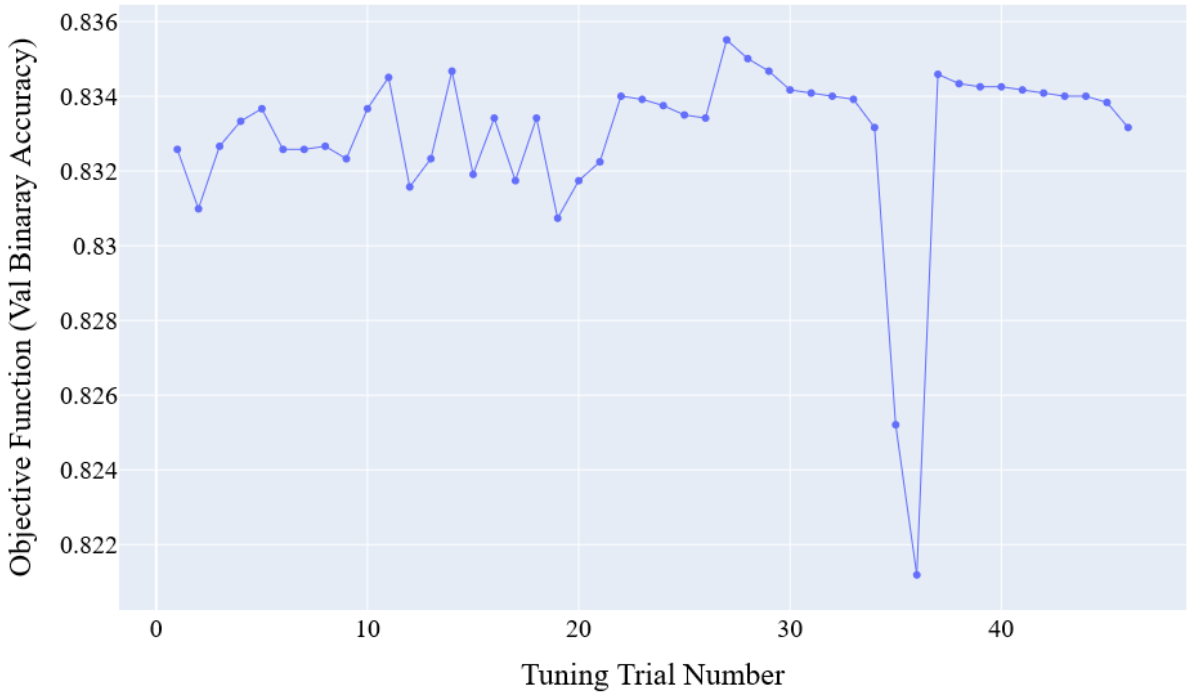


Figure 3.8 Objective function value variation during tuning job

Then [figure 3.9-3.12] contains dynamic graphs generated using Plotly library where they visualize the Mean Squad Error (MSE) and binary accuracy variation during the training of every tuner trails. By observing these four graphs it's possible to identify the hyperparameter configurations where MSE and binary accuracy is unstable against training epochs.

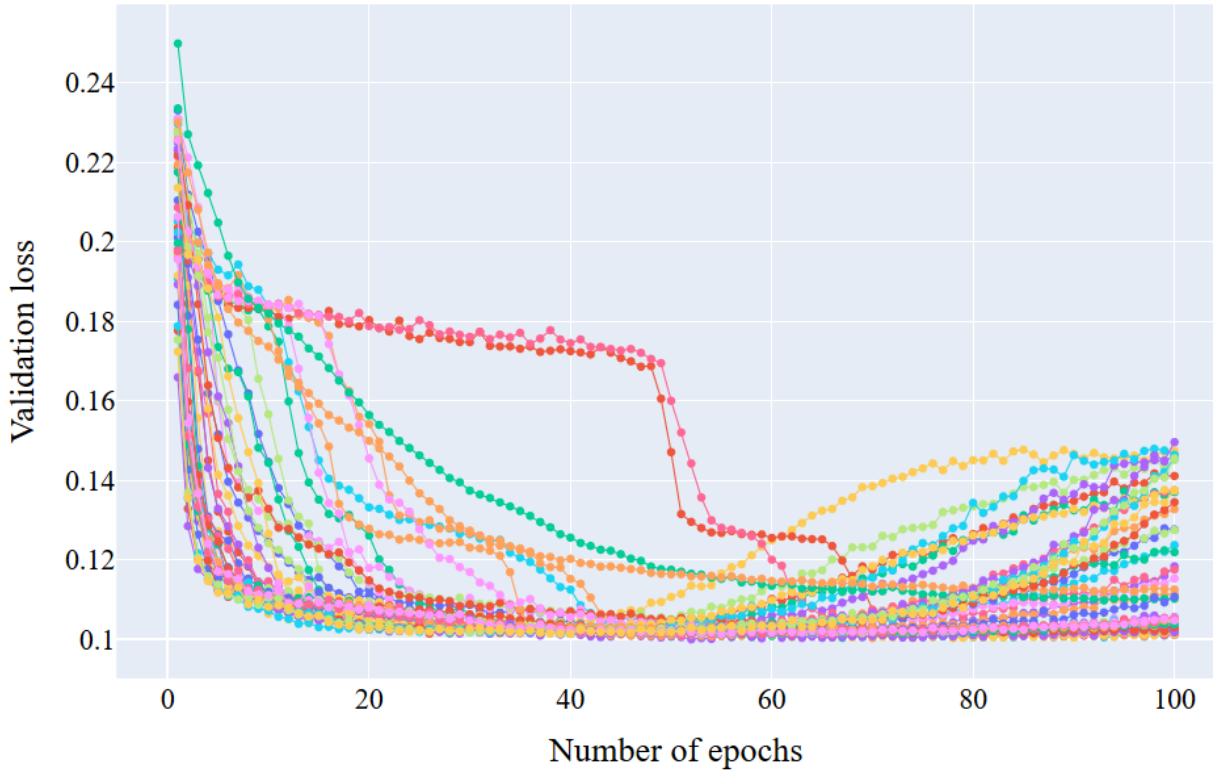


Figure 3.9 Validation Mean Squad Error (MSE) of tuner trials

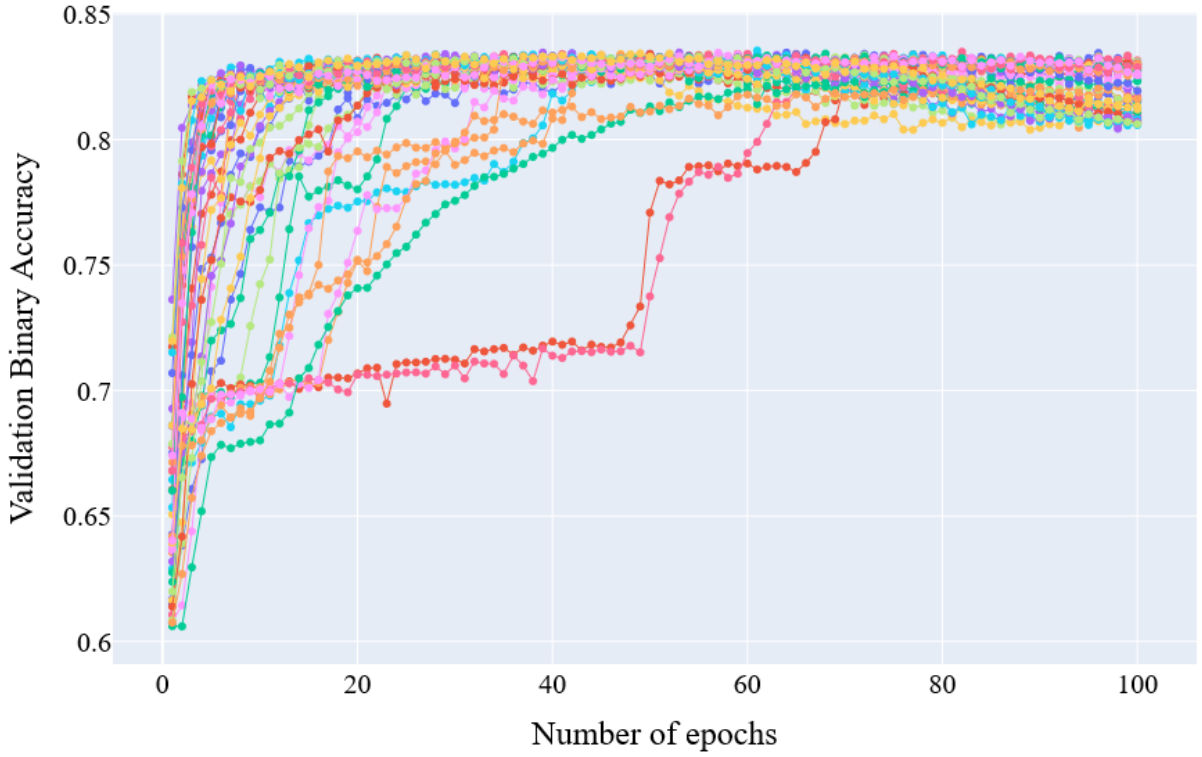


Figure 3.10 Validation Binary Accuracy of tuner trials

[figure 3.9] and **[figure 3.10]** shows the model validation results of LSTM tuner trails based on the test dataset. Analyzing these figures, it is possible to notice when optimizer select total number of LSTM units of trials larger than the total number of LSTM units of initial model (60 LSTM units), MSE loss and binary accuracy converged to maximum values within first 40 epochs. Form the other hand, when optimizer selects the total number of LSTM units of trials smaller than the total number of LSTM units of the initial model, MSE loss and binary accuracy takes more than 40 epoch to get converged to their maximum values. With this, it's possible to select an appropriate set of hyperparameters for faster model training.

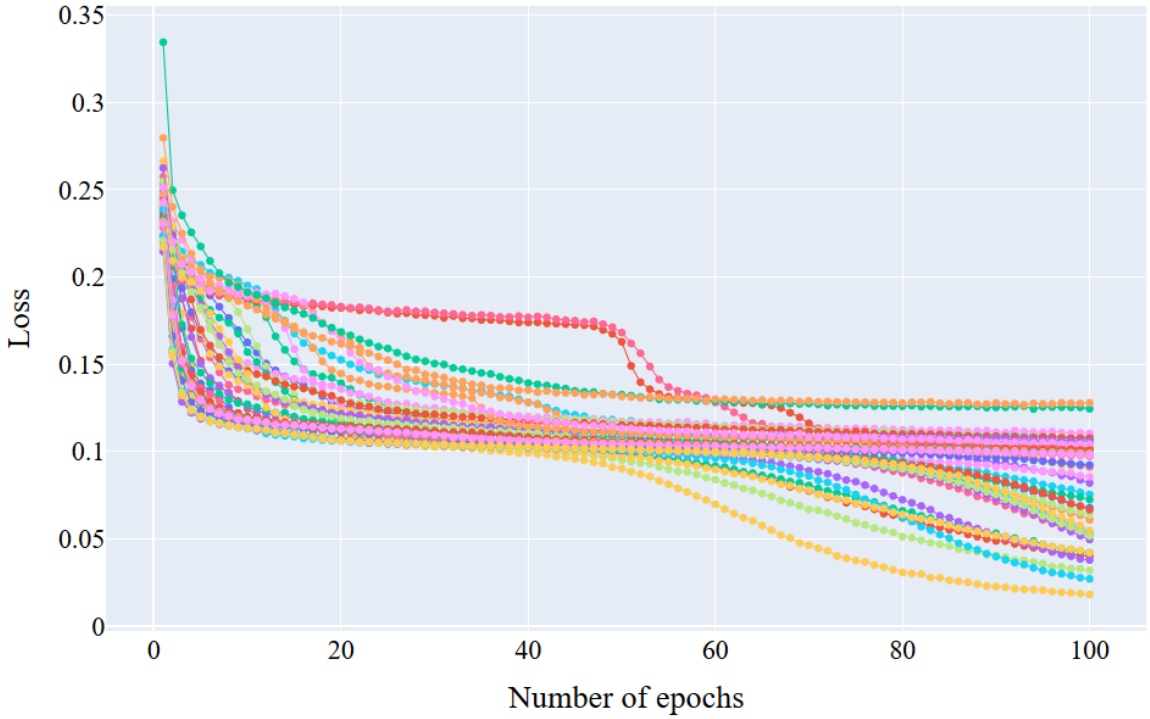


Figure 3.11 Mean Squad Error (MSE) of tuner trials

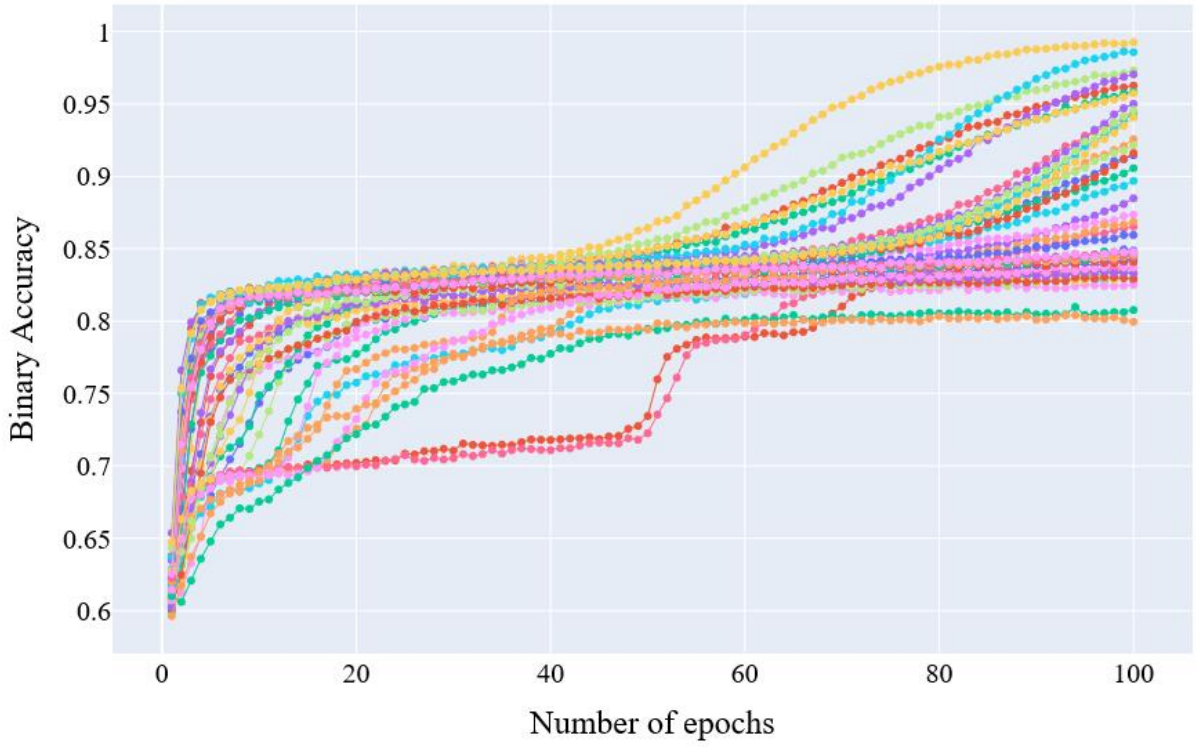


Figure 3.12 Binary Accuracy of tuner trials

Furthermore by comparing **[figure 3.10]** with **[figure 3.12]** it's possible to identify when optimizer set trial's total number of LSTM units smaller than 60 LSTM units, there are overfitting issues on training dataset where binary accuracy calculated on training dataset increase almost close to 100% while validation accuracy remains around 83%.

[Appendix-E] section 6 includes Python codes use to analyze the statistical parameters of objective function values of tuner trials using groupby and describe Pandas data analysis methods. [Table 3.2] gives the results of this analysis where maximum mean objective function values achieved when there are no hidden layers. Second column trial counts are the number of time optimizer has select particular hyperparameter value during the tuning job. After that, the minimum, mean and maximum objective function values for the different number of hidden layers and input layers units have been stored as NumPy arrays.

Table 3.2 Statistical parameters of objective function values of tuner trials

Number of hidden layers	Trial count	Statistical Parameters of objective function value (Binary Accuracy)						
		Mean	std	Min	25%	50%	75%	Max
0	20.0	0.833082	0.003482	0.821189	0.833899	0.834087	0.834276	0.835511
1	13.0	0.833327	0.000885	0.831742	0.832580	0.833417	0.833920	0.834673
2	4.0	0.832747	0.001288	0.830988	0.832182	0.833124	0.833689	0.833752
3	9.0	0.832189	0.000799	0.830737	0.831742	0.832328	0.832663	0.833501

After that under section-7 of **[Appendix-E],** first scatter plot of objective function variation against number of input layer units and number of hidden layers has obtained. After that, polynomial regression model has built to obtain the best fitting line for the scatter plot. **[Figure 3.13]** shows the objective-function value variation against number of input layer units where the objective function value reach to maximum around 100 input layer units.

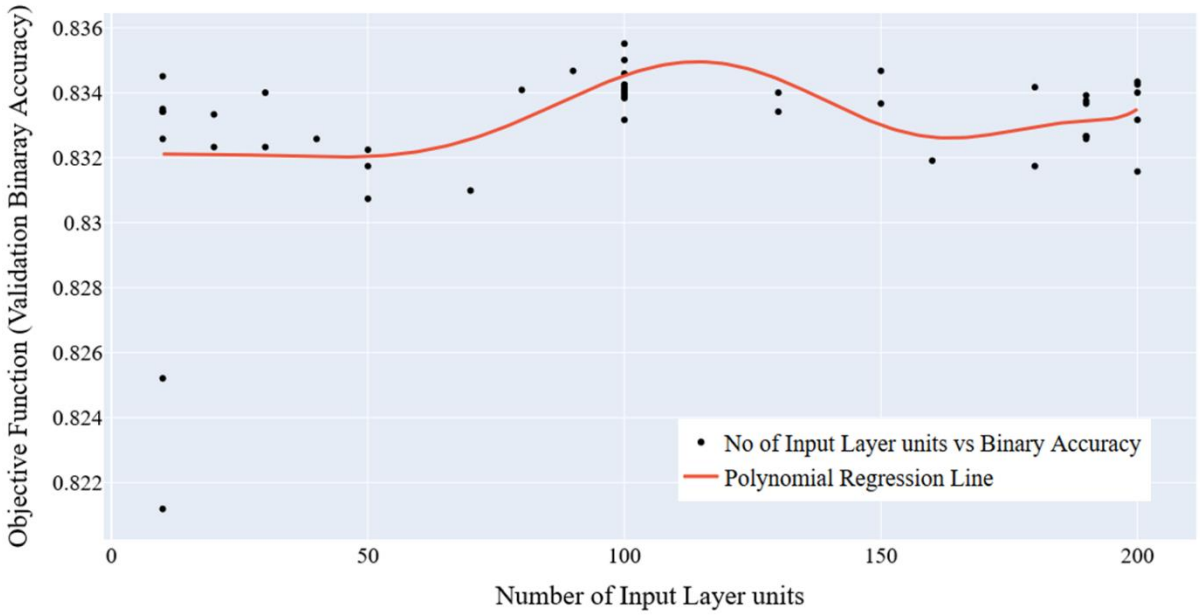


Figure 3.13 Variation of Binary Accuracy against Number of Input Layer units

A similar way **[Figure 3.14]** shows the objective-function value variation against the number of hidden layers where the objective function value reaches a maximum when there is a single hidden layer.

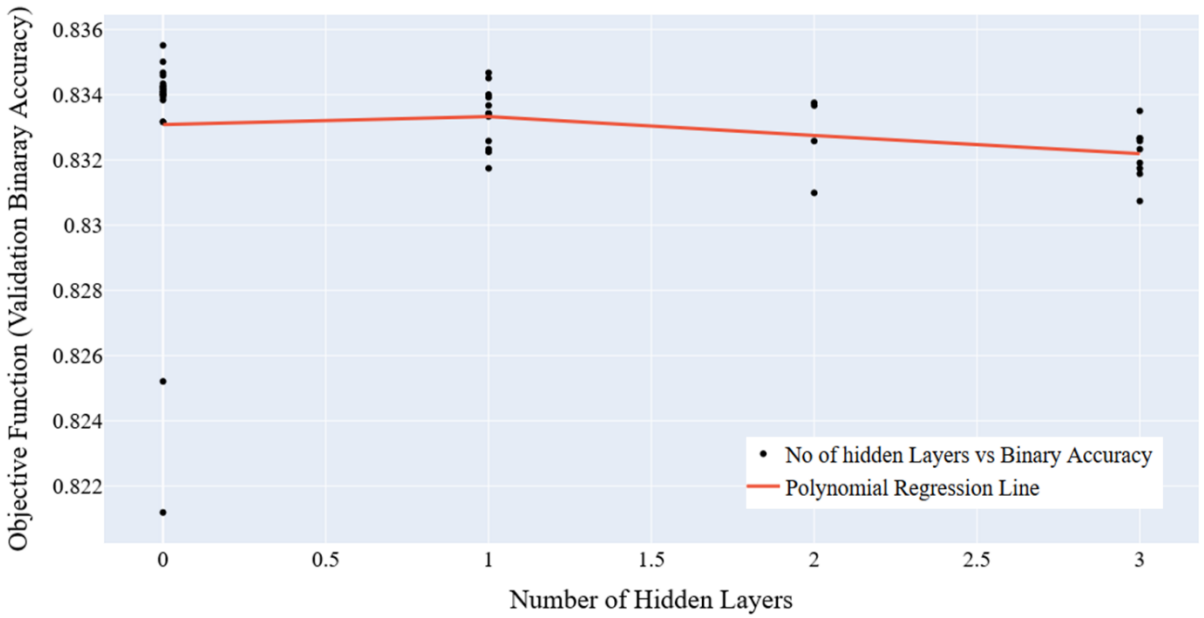


Figure 3.14 Variation of Binary Accuracy against Number of Hidden Layers

Secondly, to obtain the optimal restructured sequence length another tuning job done using AWS automatic model-tuning feature and Jupyter notebook associated with the task include under **[Appendix-F]**. Similar way after obtaining tuning data and logs HPO result analysis was carried out and Jupyter notebook associated with that includes under **[Appendix-G]**. Variation of sequence length with binary accuracy is given in **[figure 3.15]**. The bayesian optimizer has successfully identified configurations where the model could achieve an accuracy of around 83% for almost all the lengths. But a slight increment in accuracy is obtained when the sequence is length is increased. This could be because of the additional information contributed by the long sequence length which is stored in LSTM cells. However, it is safe to say that, despite the length of each restructured sequence, there is a possibility of achieving the best prediction accuracy if the rest of the parameters are adjusted

accordingly. These parameters can be selected based on factors such as computational power required and memory constraints.

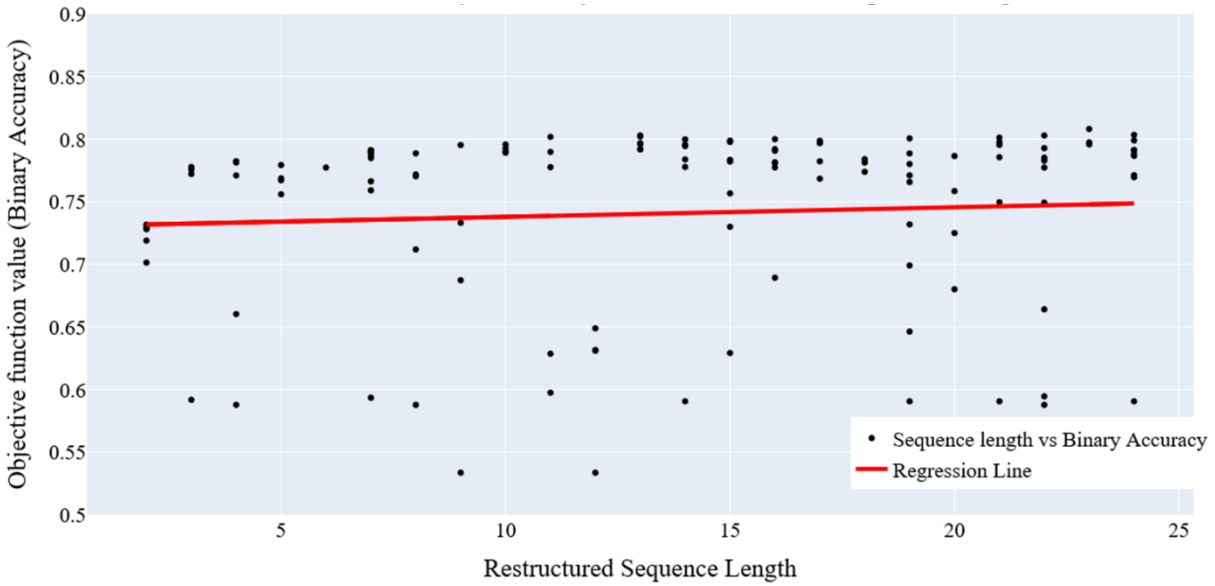


Figure 3.15 Variation of Binary Accuracy against Sequence length

By considering above all analysis and tuning trial with the highest objective function values, results obtained under this tuning jobs is summarized in **[Table 3.3]** with ranges of layer cell numbers used.

Table 3.3 Hyper-parameter ranges used and best model parameters

Hyper parameter	Range	Best Model Parameters
Number of input layer units	10-200	100
Number of hidden layers	0-3	0
Number of hidden layer units	10-200	0
Batch size	32-128	113
Dropout	0.1-0.9	0.27
Learning rate	0.001-0.1	0.001
Loss type	Mean Squared Error, Logcosh	Mean Squared Error
Optimizer	Adam, RMSprop	Adam
Number of Epoch	1-50	26
Number of dense layer units	-	5
Objective function	-	Binary Accuracy

After both tuning jobs and results analysis, the best model has trained with table 3.3 tuning results. A comparison of convergence in binary accuracy and MSE of the initial model and tuned model is given in **[figure 3.16 and figure 3.17]**. The tuned model is faster during the training process and converges to a higher accuracy level at the end of a similar number of epochs as the initial model.

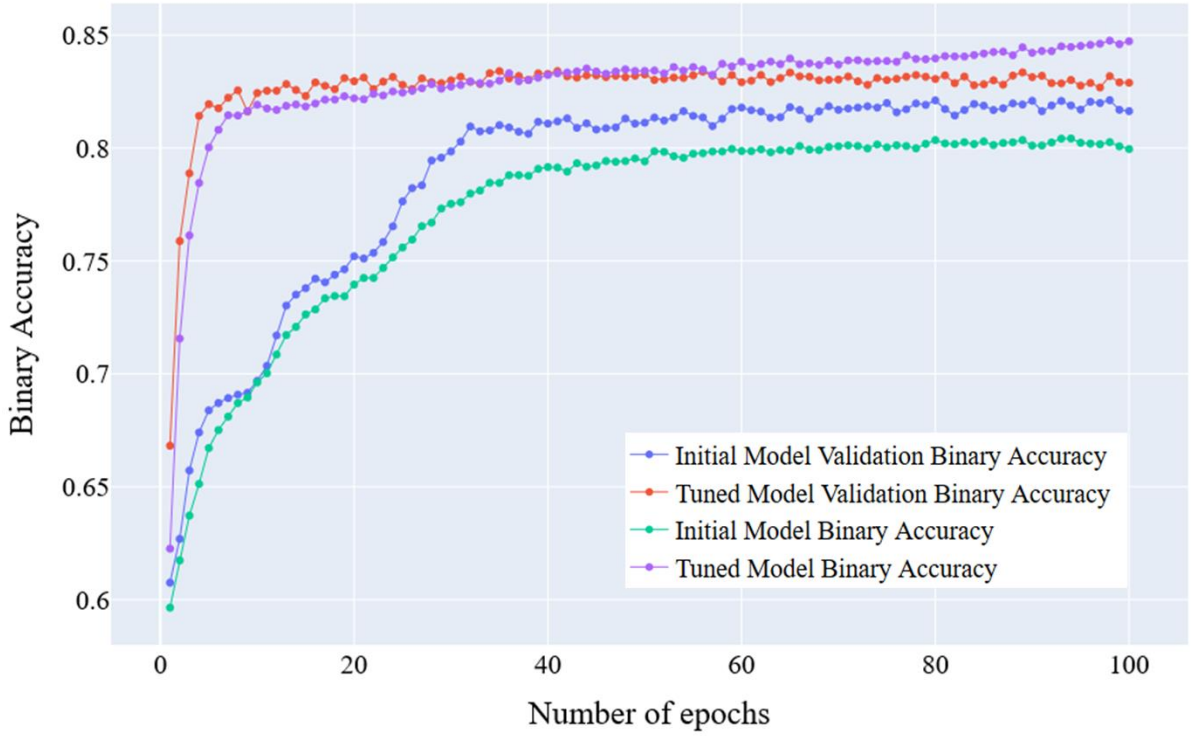


Figure 3.16 Comparison between initial model and tuned model binary accuracy variation during training process

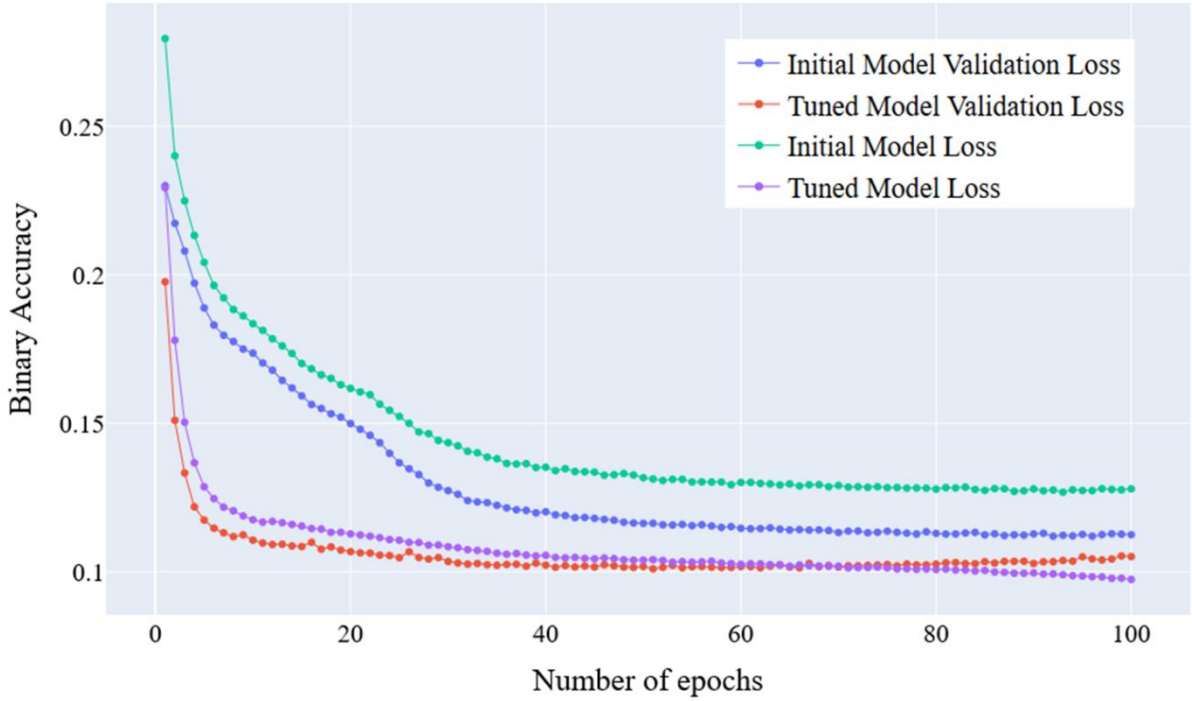


Figure 3.17 Comparison between initial model and tuned model Mean Squad Error (MSE) variation during training process

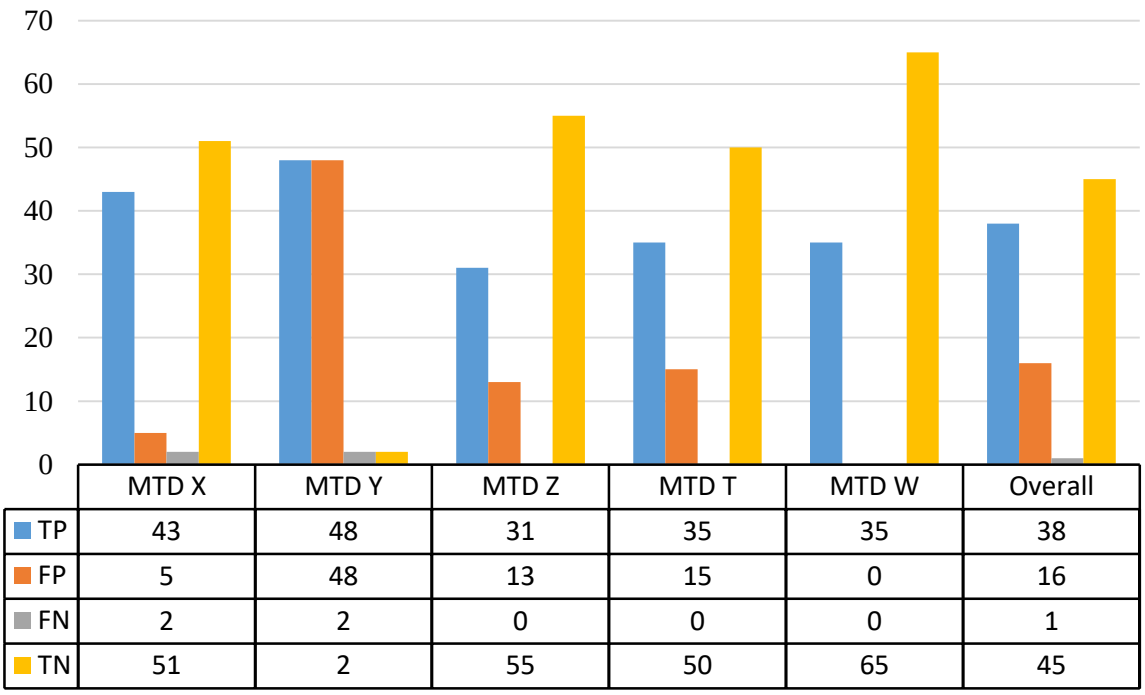
3.1.5 LSTM Model Performance Evolution

The results obtained from the model gives us whether the devices in the MTD network will transmit or not during the future timesteps. In other words, after normalize and round-up final LSTM of output values takes finery form. The confusion matrices for each device along with the overall confusion matrix of the system is given in **[Table 3.4].**

Table 3.4 Confusion Matrices of MTDs and the System

	TP	FP	FN	TN	Total
MTD X	1024	111	46	1207	2388
MTD Y	1140	1143	46	59	2388
MTD Z	752	318	0	1318	2388
MTD T	839	348	0	1201	2388
MTD W	839	0	0	1549	2388
Overall	4594	1920	92	5334	11940

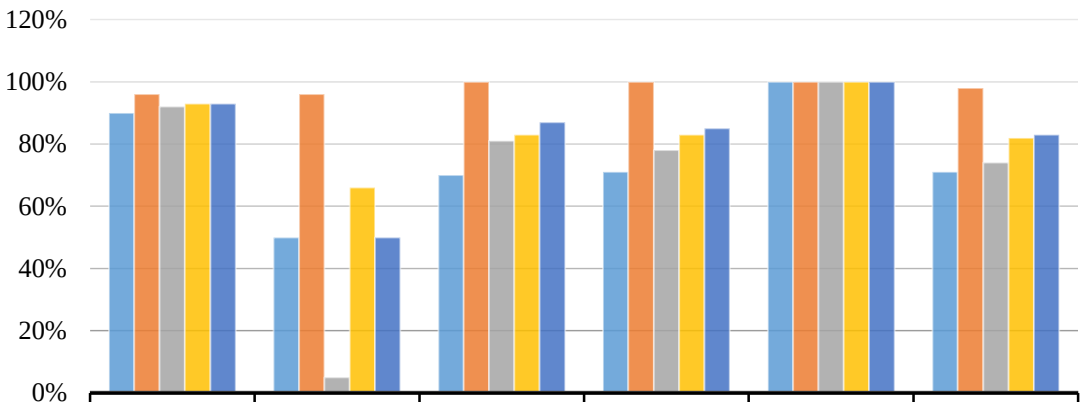
[Figure 3.18] shows the True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN) as a percentage where TP and TN consider as correct predictions and FN and FP consider as incorrect predictions.



	MTD X	MTD Y	MTD Z	MTD T	MTD W	Overall
TP	43	48	31	35	35	38
FP	5	48	13	15	0	16
FN	2	2	0	0	0	1
TN	51	2	55	50	65	45

Figure 3.18 Confusion matrix values as percentage

According to the results show in **[figure 3.19]** that accuracies for devices X, Y, Z, T, W of the LSTM model are 93%, 50%, 87%, 85%, 100%. It is seen that the devices which are causally dependent on other devices' transmissions have higher prediction accuracies than event-initiating devices. The overall prediction accuracy of the system is 83%.



	MTD X	MTD Y	MTD Z	MTD T	MTD W	Overall
Precision	90%	50%	70%	71%	100%	71%
Sensitivity	96%	96%	100%	100%	100%	98%
Specificity	92%	5%	81%	78%	100%	74%
F-score	93%	66%	83%	83%	100%	82%
Accuracy	93%	50%	87%	85%	100%	83%

Figure 3.19 LSTM Model performance evaluation parameters

Therefore, LSTMs can be viewed as a promising approach for predicting source traffic prediction in a Fast Uplink Grant environment. Furthermore, apart from better metrics such as accuracy, LSTMs have an added advantage in scalability. They have the capability to handle any number of sequences from each MTD, essentially when IoT networks grow in size.

3.2 Extended LSTM Network Implementation

In section 3.1, a very small cellular-enable MTC network with 5-MTDs considered keeping the system simple for analytical purposes. However, as the second stage of the Fast Uplink Grant concept that is optimal channel allocation, it required an MTC network with a large number of MTDs for interpretable analysis in terms of MTD latency requirements. Therefore, to address this analytical requirement of Fast Uplink Grant allocations the LSTM based source traffic prediction framework scaled up for enlarged MTC network with 100 MTDs. One's MTC network simulation done for with 100 MTDs reset of LSTM implementation steps follows similar to previous LSTM network implantation described in section 3.1. Therefore, under this section very brief detail of LSTM model building, training, tuning, and performance evaluation have presented.

3.2.1 Extended MTC Network Simulation

Extended MTC network simulation programme created to generate binary transmission time series for 100 MTDs where 5-MTDs exhibits periodic transmissions, other 5-MTDs exhibits random transmissions and rest of MTDs exhibits event-driven transmissions. Every event-driven transmitting MTD have a causal relationship with periodic transmitting MTD or random transmitting MTD as shown in **[figure 3.20]**. The periodic transmitting MTDs and random transmitting MTDs act as event triggering MTD. Therefore, we assume whenever IoT event occurred this event triggering MTD get affect by the event before other MTDs exhibits event-driven transmission. This assumption hold because of the geographically propagating nature of IoT events.

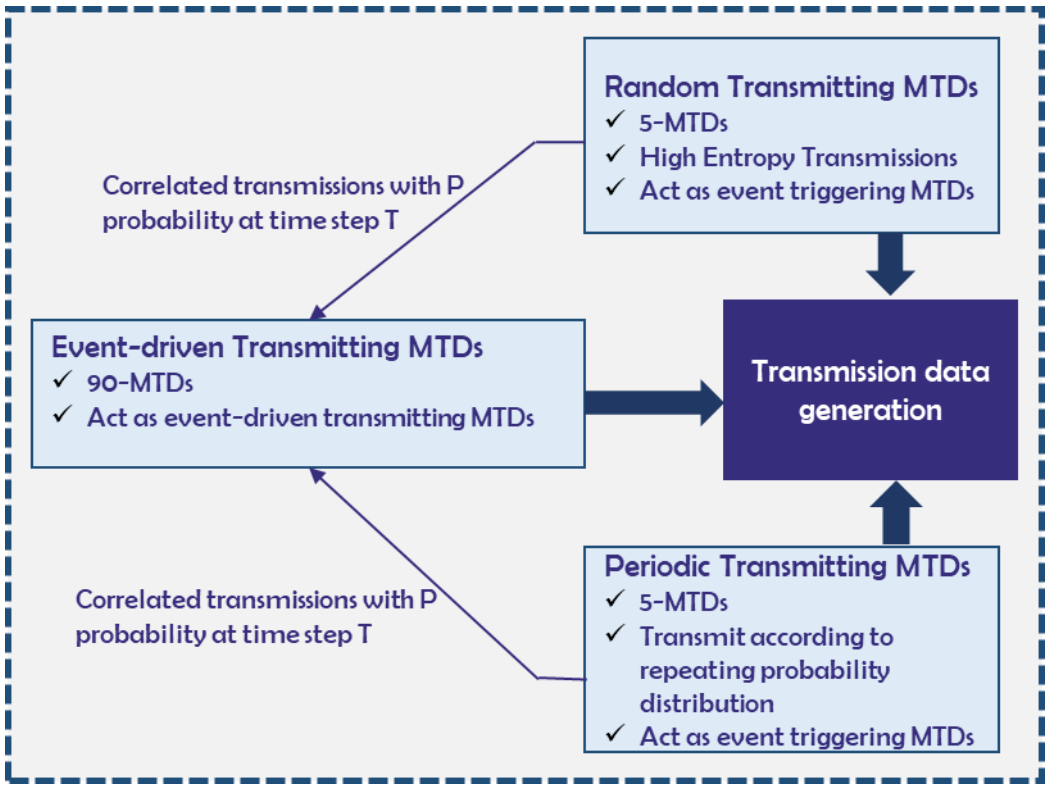


Figure 3.20 Enlarged MTC Network Simulation

Periodic transmission happens based on fixed repeating probability distribution where transmission probability is periodically set to a value within interval [0, 1] throughout the transmission time series. As an example transmission probability of time step **(10N +X)** is set to fix **P** probability where **N** is natural number and **X = [0, 9]** and **P = [0, 1]**. Random transmission can be happen at any given time step complete randomly. In other work for any given time step probability of transmission is 0.5 and probability of being silent is 0.5. Event-driven transmission occur with **P** correlated probability to periodic and random transmitting MTD after **X** time step to event is triggered where **P = [0.8, 1]** and **T = [0, 9].** Python codes associated to the MTC network simulation included to **[Appendix-I]**. Finally all the 100-MTD transmission stack in to single array of (100, 60000) shape using **numpy.vstack** function.

3.2.2 Extended LSTM Model Implementation

Similar to previous LSTM implementation before data feeding to the LSTM, the generated data is reshape most optimally for the model to understand. In this LSTM implementation next single time step after the first 10, is taken as the labels in the generated transmission data. They are given as 10th, 11th, 12th, etc in **[figure 3.21]** for each MTD. During the training process, the transmission pattern of the previous 10 time steps is fed to the LSTM as the independent feature matrix alongside with the labels.

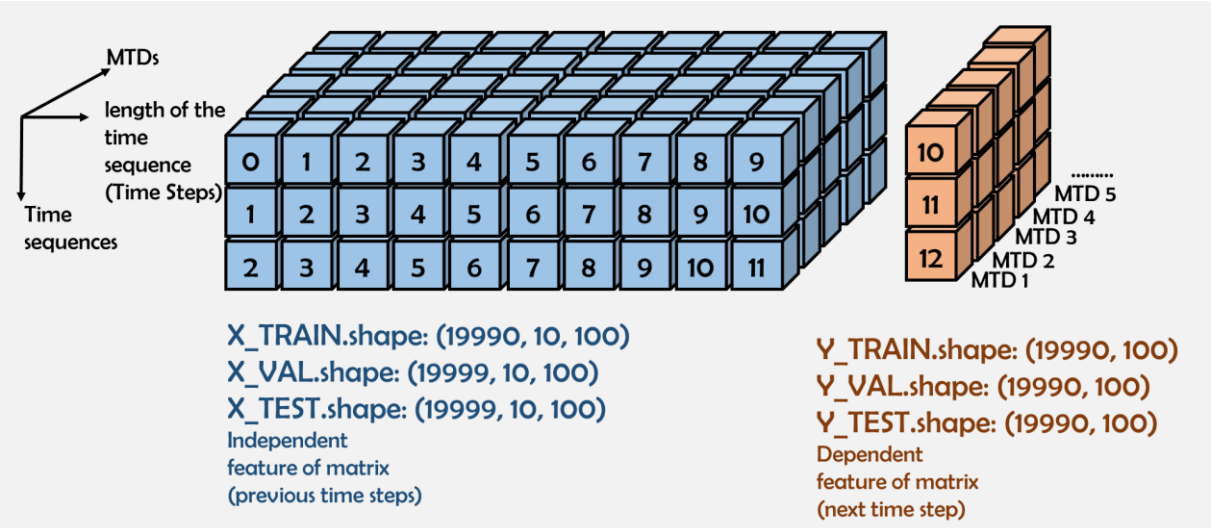


Figure 3.21 LSTM input data Structure

Python codes use for data preprocessing included in **[Appendix-I]**. First transmission data of 60000 time steps split into training, validation and testing datasets where each dataset has transmission data of 20000 time steps. Then NumPy inbuilt array manipulation functions such as reshape, append, delete and indexing apply on (100, 20000) shape training, validation and testing datasets to get independent and dependent feature of the matrix with the shape of (19990, 10, 100) and (19990, 100) respectively. Finally, all 4 matrix save as NPZ files by **np.savez** function.

After similar to previous implantation, LSTM model was first trained with arbitrary parameters and then it was trained using a Bayesian Optimizer. Initially build LSTM model structure and the layer sizes were obtained based on the literature **[Learning capability and storage capacity of two-hidden layer feedforward networks]** where the first layer and hidden layer cell numbers were taken using an equation. The rest of the parameters are added arbitrarily or according to the general convention.

LSTM model tuning used for determining optimal LSTM cell numbers for the input layer and hidden layers to obtain the best validation binary accuracy. By considering HPO tuning job analysis and tuning trial with the highest objective function values, results obtained under this tuning job is summarized in **[Table 3.5]** with ranges of layer cell numbers used.

Table 3.5 Hyper-parameter ranges used and best model parameters

Hyper parameter	Initial Model Parameters	Range	Best Model Parameters
Number of input layer units	520	50-1000	150
Number of hidden layers	2	0-3	2
Number of hidden layer units	300	50-1000	300

Finally, the best model has trained with **table 3.5** tuning results. A comparison of convergence in binary accuracy and MSE of the initial model and tuned model is given in **[figure 3.22 and figure 3.23]**. The tuned model is faster during the training process and converges to a higher accuracy level at the end of a similar number of epochs as the initial model.

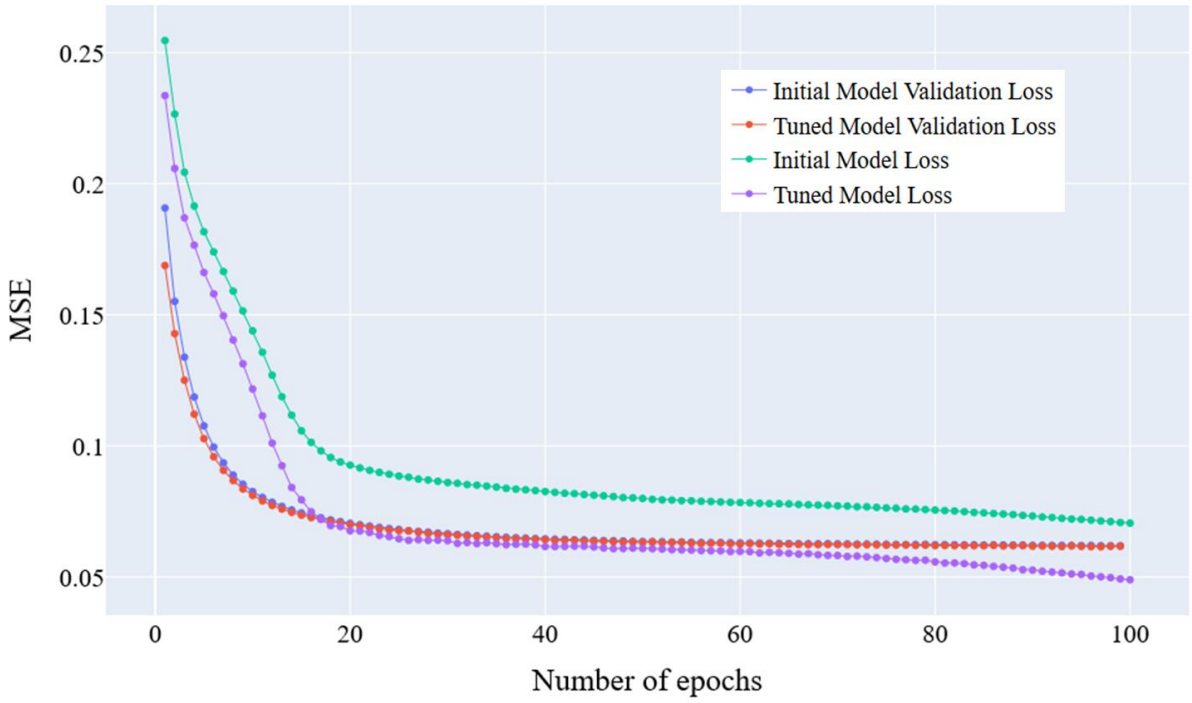


Figure 3.22 Comparison between initial model and tuned model Mean Squad Error (MSE) variation during training process

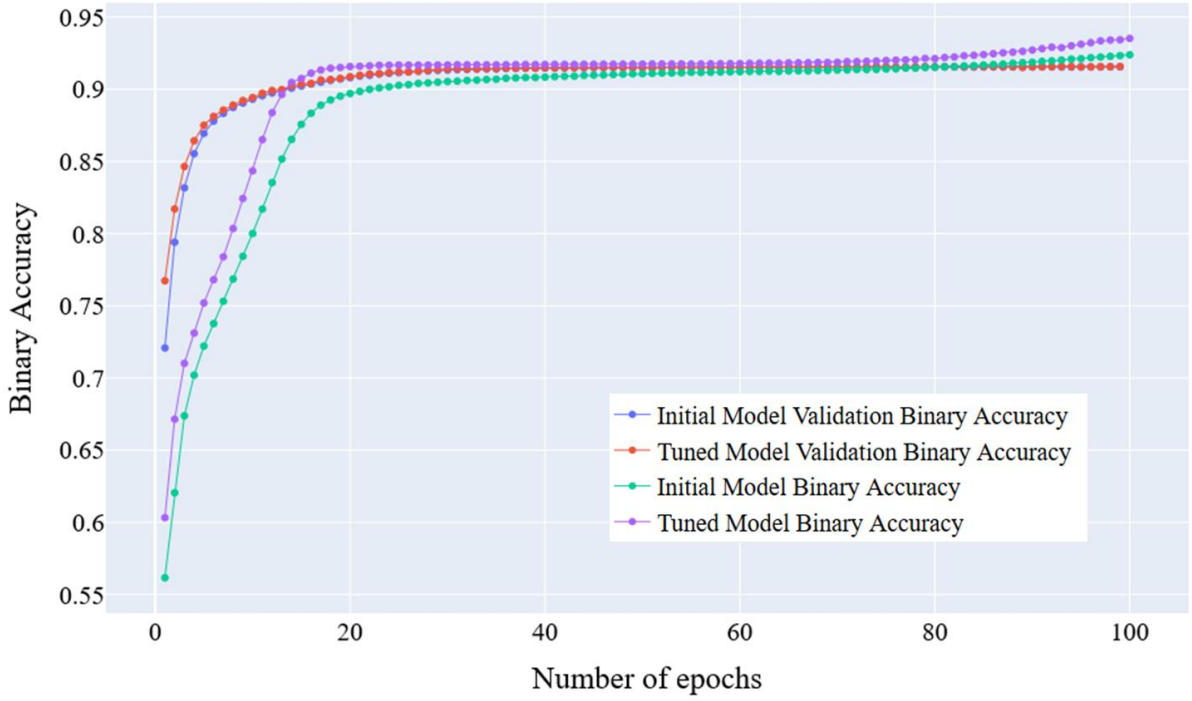


Figure 3.23 Comparison between initial model and tuned model binary accuracy variation during training process

[Figure 3.24] shows overall LSTM performance matrix values such as TP, FP, FN, TN, precision, sensitivity, specificity, f-score, FDR and accuracy as a percentage where TP and TN consider as correct predictions active MTDs and sleeping MTD respectively. A definition of these performance matrixes included in **[Appendix-J]** under define model performance evaluation function.

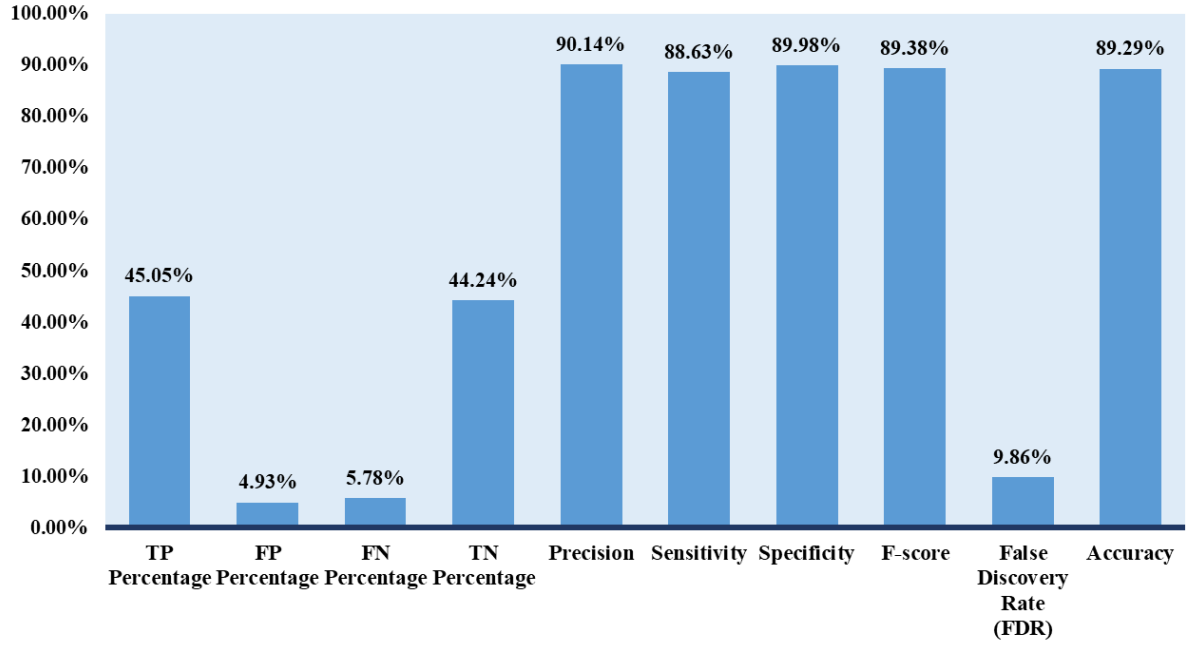


Figure 3.24 Overall LSTM performance matrix values

[Figure 3.25] shows prediction accuracy of each MTD where prediction accuracy of random transmitting 5-MTDs are around 50% and other all MTD has achieved more than 70% prediction accuracy.

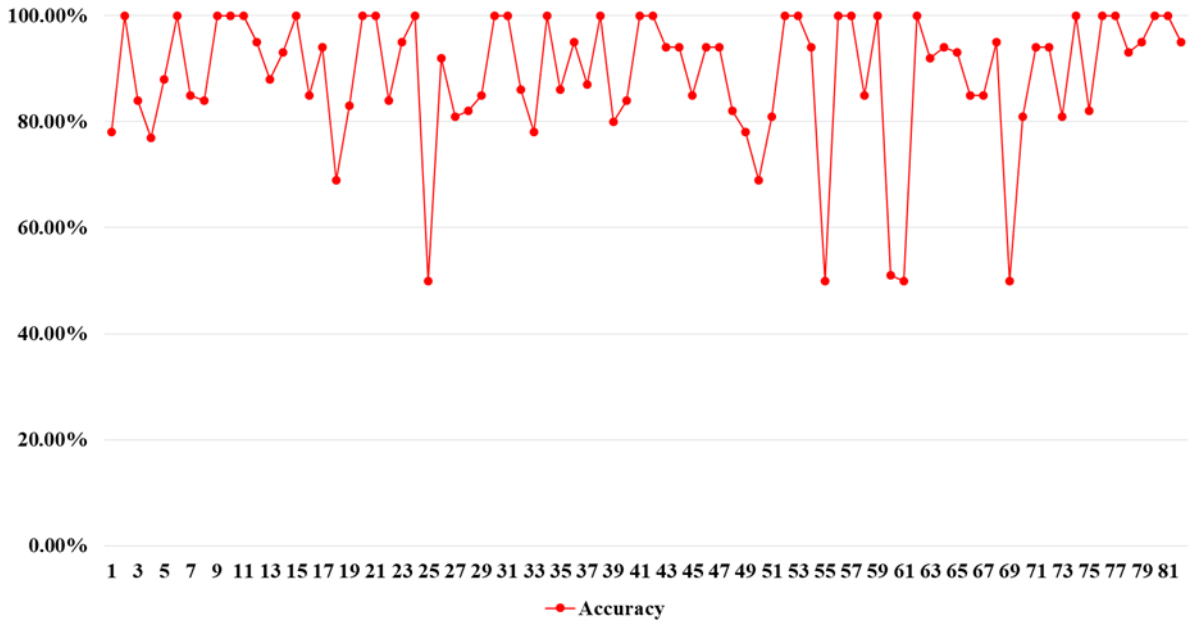


Figure 3.25 Individual prediction accuracy of MTD in the network

3.3 Optimal Fast Uplink Grant Allocations

As mentioned in the literature review the second stage of the Fast Uplink Grant concept is the optimal allocation of the Fast Uplink Grant considering the QoS requirements. Existing research has proposed a Probabilistic Multi-Arm Bandit Theory for this purpose. In this study, the Probabilistic Sleeping MAB algorithm was implemented with results from LSTM based traffic prediction model.

As illustrated in **[Figure 3.26] c**onsider an uplink channel resources of a cellular-enable MTC network composed of one BS and M number of MTDs that use a Fast Uplink Grant. The BS uses the proposed LSTM based model for the source traffic prediction stage of the Fast Uplink Grant. Scheduling is also done at the base station. It is assumed that the total available bandwidth is divided into resource blocks of each of which is of size W and *T.*

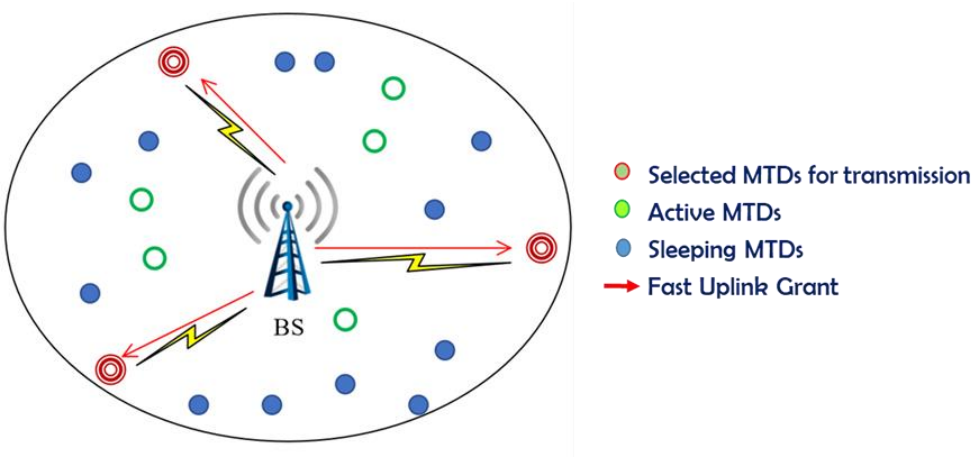


Figure 3.26 Fast Uplink Grant enabled MTC Network

The latency requirements of MTDs represented as maximum tolerable access delay di(t). It is important to note that $d_i(t)$ is a total delay that can be tolerated from the time instance *t* at which the data packet is ready to be transmitted at the MTD queue until it is scheduled to be sent. Queuing, transmission and propagation delays are assumed to be already deducted from the total latency requirements of each IoT application that has generated the data packets. Maximum tolerable access delay of each MTD changes over time since at different times, data packets might have had different queueing times, or even different latency requirements due to various QoS requirements of MTC applications that are generating the packets.

The proposed probabilistic multi-armed bandit based allocation policy was implemented by considering both LSTM predictions and perfect predictions (real transmission data in the test dataset) as MTD availabilities to investigate the effect of prediction on the channel allocations. A random allocation policy was implemented to compare the performance of the proposed method. The proposed optimal fast uplink grant allocation mechanism can be graphically represented using a block diagram in the **[figure 3.27]**.

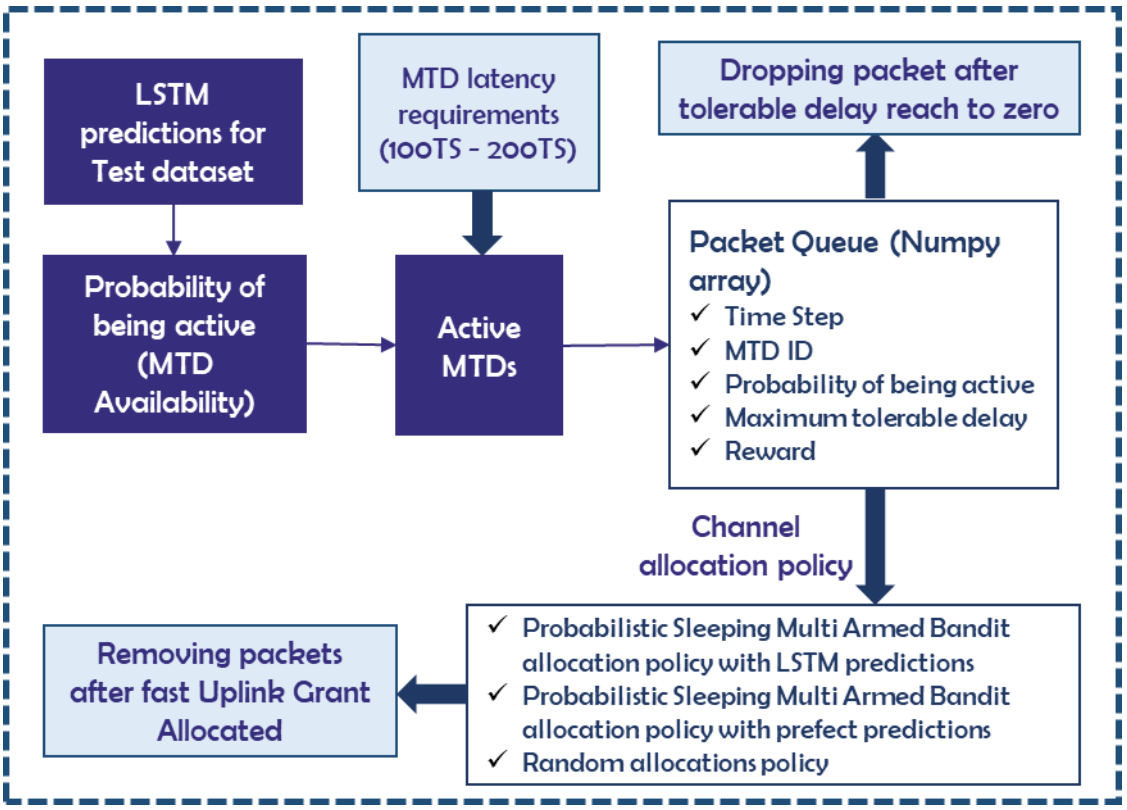


Figure 3.27 Fast Uplink Grant Allocation Framework

3.3.1 MTD Availability and Latency Requirements

Active MTDs or available MTDs for data transmission for a given timestep determined by the LSTM based prediction model proposed by this study. To investigate the effect of prediction error for allocation, MTD availability considers two different methods for the proposed MAB based allocation policy. The first method was to get the probability of being active from normalized output LSTM predictions. The second method to get the MTD availability directly from a real test set transmission data by considering the prefect prediction scenario.

If Pi(t) > 0.5, the MTD was considered to be active and has data to transmit during the particular timestep. The proposed LSTM model uses to generate predictions for test datasets with 20000 timesteps. The total number of active or available MTD for given timesteps can be varied as shown in **[figure 2.28]** for predicted timesteps. As a results of the statistical analysis about MTD availability **[figure 2.28]** generated and based on that we have select number of channel for this optimal Fast Uplink Grant allocation framework. Therefore, for this particular dataset we have consider 20 uplink traffic channel where 20 MTDs was selected for each timestep by proposed probabilistic MAB allocation policy.

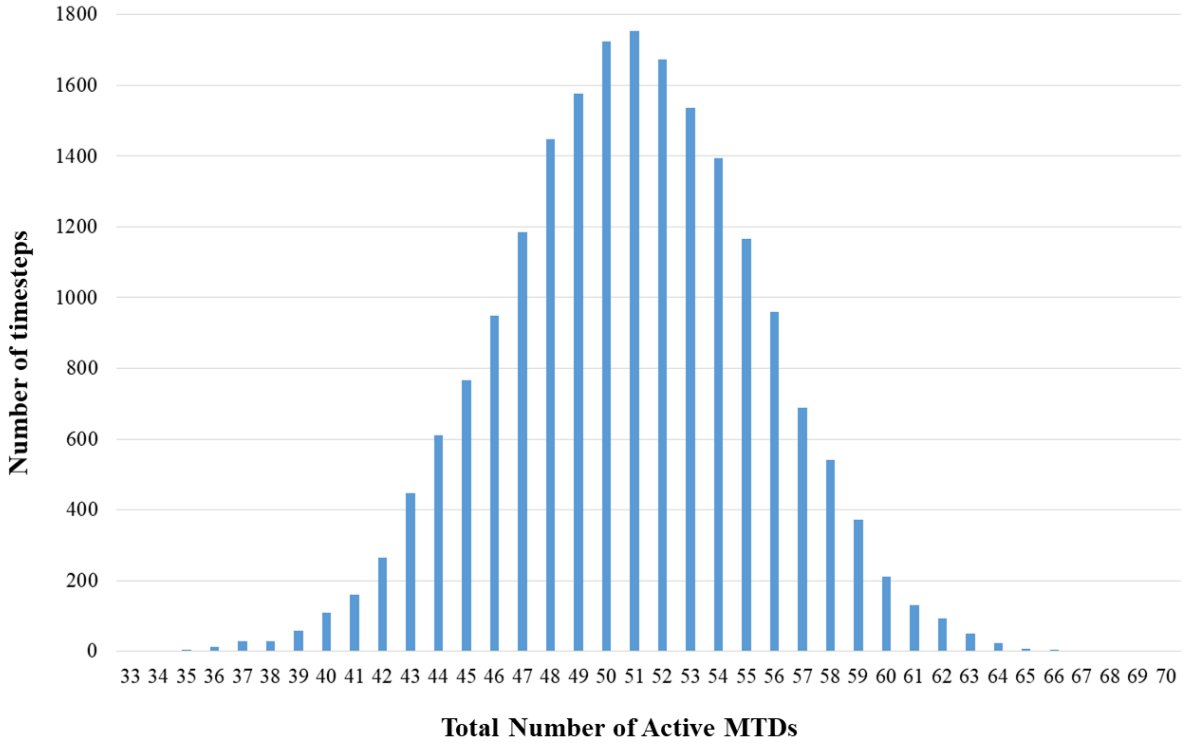


Figure 3.28 Variation of total number of active MTDs within LSTM prediction dataset

Each data packet was assigned a value between 100 to 200 timesteps as its maximum tolerable delay where that value gradually reduced while the data packet is waiting for uplink grants in the MTD's queue. The summary of the latency requirements of the MTC Network shown in **[figure 2.29].**

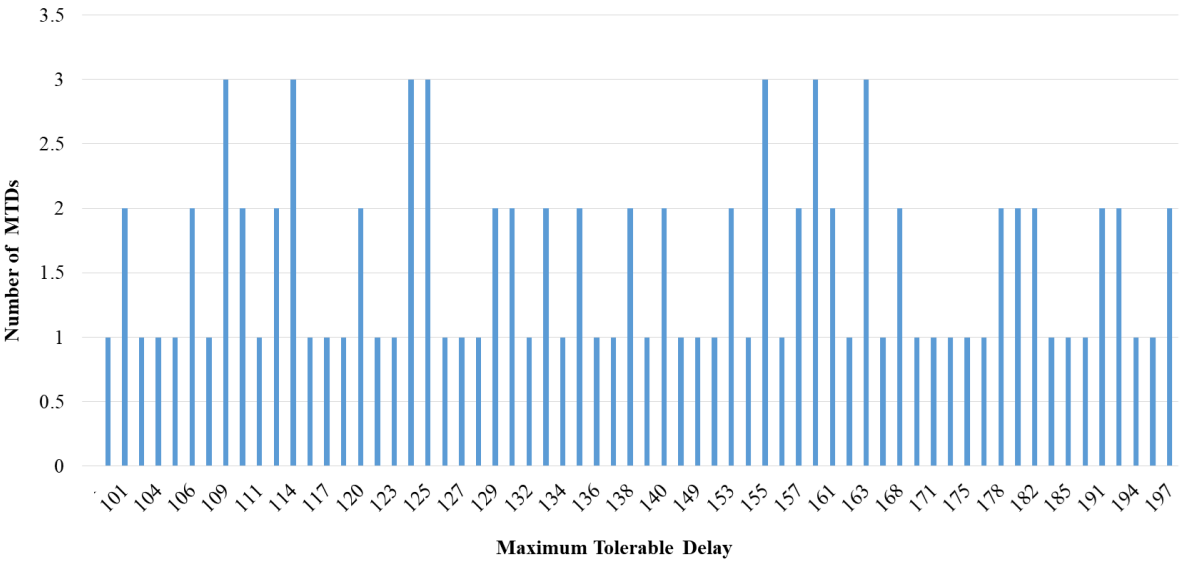


Figure 3.29 Latency Requirements of MTDs

3.3.2 MTD Latency Requirement based Reward Function

The first step of this implementation is to map the maximum tolerable access delays to the number between **0 -1**. To achieve this the modified Gompertz function given in **[Equation]** was used and **a = 1**, **b = 14**, and **c = 0.025** was used. By following literature lower maximum tolerable access delay values were mapped to higher rewards while higher maximum tolerable access delay values were mapped to lower rewards. **[Figure 3.30]** illustrates the curve obtained for the maximum tolerable access delays of the dataset,

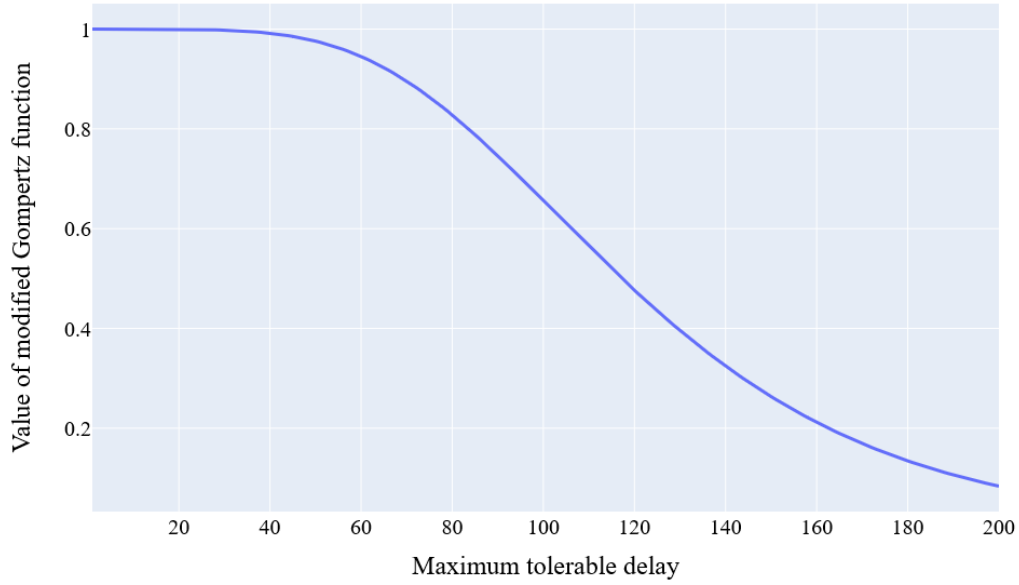


Figure 3.30 MTD latency requirement based reward function

The channel rate information and value of information were neglected for this analysis, as the target was studying the performance in terms of latency. Hence the values of reward utility function **[alpha], [beta]** were set to zero and the value of **[gamma]** was set to 1 so that reward now only depends on the maximum tolerable delay.

3.3.3 Packet Queuing Mechanism

According to the optimal grant allocation literature, MTD packets queuing mechanism implemented and whenever MTD packets become available due to having transmission probability **[Pi(t)]** higher than 0.5, that particular MTD packets are added to the queue. The packets will be a drop from the queue, if only they are selected to receive the Fast Uplink Grant or if they had to wait to more than maximum tolerable access delay to receive the Fast Uplink Grant. The following **[figure 3.31]** shows the content of the implemented MTD packet queue. Furthermore, after passing each timestep tolerable delay decreased by one timestep and reward is update accordingly while other columns' values remain unchanged.

Timesteps index **[n]**	MTD index **[i]**	Probability of being Active **[Pi(t)]**	Tolerable delay **[Di(t)]**	Reward **[Ri(t)]**
19808.0	89.0	0.740901	1.0	0.999997
19811.0	89.0	0.528047	4.0	0.999992
19812.0	89.0	0.709892	5.0	0.999990
19813.0	89.0	0.646517	6.0	0.999986
19815.0	89.0	0.557743	8.0	0.999976
19817.0	2.0	0.590887	2.0	0.999996
19817.0	89.0	0.556154	10.0	0.999960
19818.0	2.0	0.721413	3.0	0.999994
19818.0	89.0	0.662792	11.0	0.999949
19822.0	2.0	0.572855	7.0	0.999982
19822.0	76.0	0.632333	27.0	0.998665

Figure 3.31 MTD Packet Queue

3.3.4 Probabilistic Multi-armed Bandit Allocation Policy with LSTM predictions

According to the proposed probabilistic Multi-Armed Bandit (MAB) allocation policy explained in the literature review, the allocation of Fast Uplink Grant for a particular timestep was done considering the multiplied value of Upper Confidence Bound (UCB) values and activation probability Pi (t) of the MTD. Here the MTD availability or activation probability gets from the output of the LSTM source-traffic prediction algorithm. The calculation of UCB value done according to the algorithm describes in section 2.5.2 and 20 uplink direction channels were consider where 20 MTDs was select for transmission in each timestep. The Following **[figure 3.32]** shows the number of times each maximum tolerable delay selected by this allocation policy.

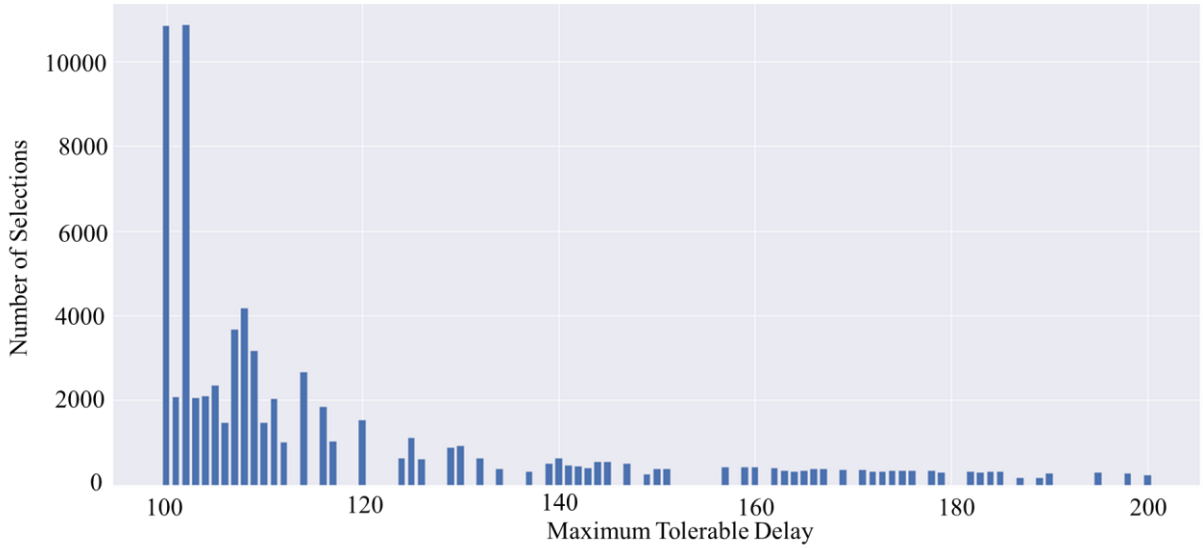


Figure 3.32 Number of times each Maximum Tolerable Delay was selected with LSTM predictions

[Figure 3.33] shows MTDs with lower maximum tolerable delay has more frequently selected over time compare to MTDs with higher maximum tolerable delay requirements. Therefore, scatter plot in [**figure 3.33]** gives clear sign that proposed MAB based allocation policy can identify the latency requirement of MTDs and perform allocation by prioritizing the delay sensitive MTDs. All the Python codes associated to implementations of this proposed allocation policy include under **[appendix-K].**

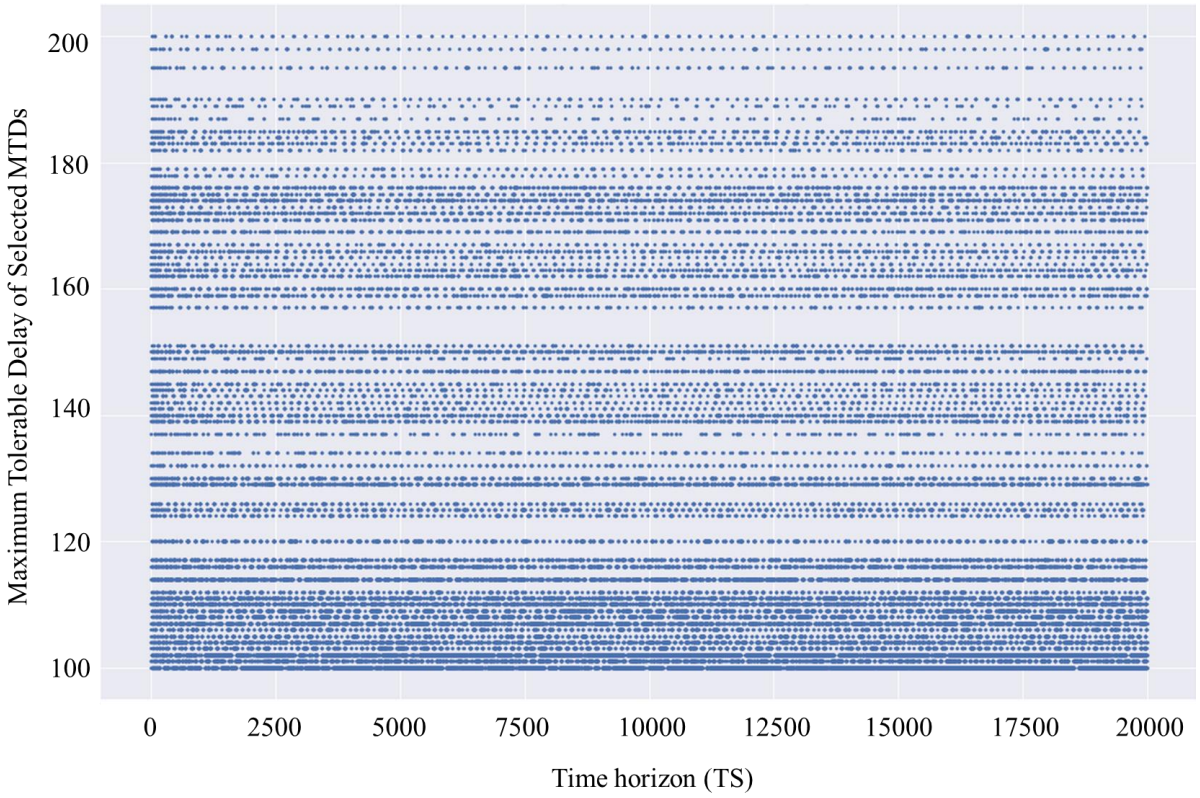


Figure 3.33 Maximum tolerable delay of the selected MTDs by MAB allocations with LSTM predictions

3.3.5 Probabilistic Multi-armed Bandit Allocation Policy with Prefect predictions

The proposed probabilistic multi armed bandit based allocation policy was implemented by considering both LSTM predictions and perfect predictions (real transmission data in the test dataset) as MTD availabilities to investigate about the effect of prediction on the channel allocations. In here MTD availabilities obtain directly from real test set transmission data by considering prefect prediction scenario. By comparing **[figure 3.32]** with **[figure 3.34]** we can see that predictions errors and the LSTM output value has case to change the number of time each MTD was selected when compare the perfect predictions scenario.

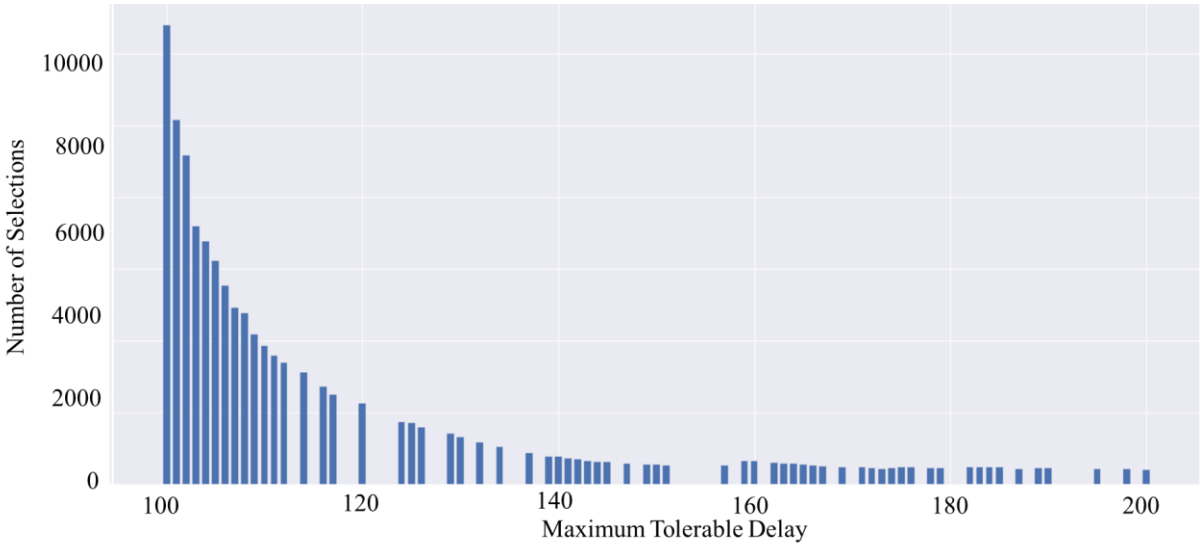


Figure 3.34 Number of times each Maximum Tolerable Delay was selected with prefect predictions

The scatter plot of the latency of the selected MTD at each time is presented in **[figure3.33 and figure 3.35]** for the proposed sleeping MAB with LSTM predictions and prefect predictions. Each dot in these

figures corresponds to the maximum tolerable access delay of the selected MTD. **[figure3.33 and figure 3.35]** shows the capability of the sleeping MAB algorithm in optimizing the latency while providing fairness in the system. From **[figure3.35]**, we can see that, in the beginning, the dots are uniformly distributed for all values of delay requirements, which means that the MTDs are randomly selected. However, after the learning, the intensity of the dots for MTDs with stricter latency requirements is much higher than that of the MTDs with larger delay requirement, which means that delay sensitive MTDs are scheduled more often. However, after learning period, the algorithm still keeps scheduling MTDs with larger latency requirements. This increases the accuracy of the information at the BS about the latency requirements of all MTDs and also provides fairness. Moreover, if the latency requirements of an MTD has changed over time, the algorithm can discover that and start scheduling that MTD accordingly.

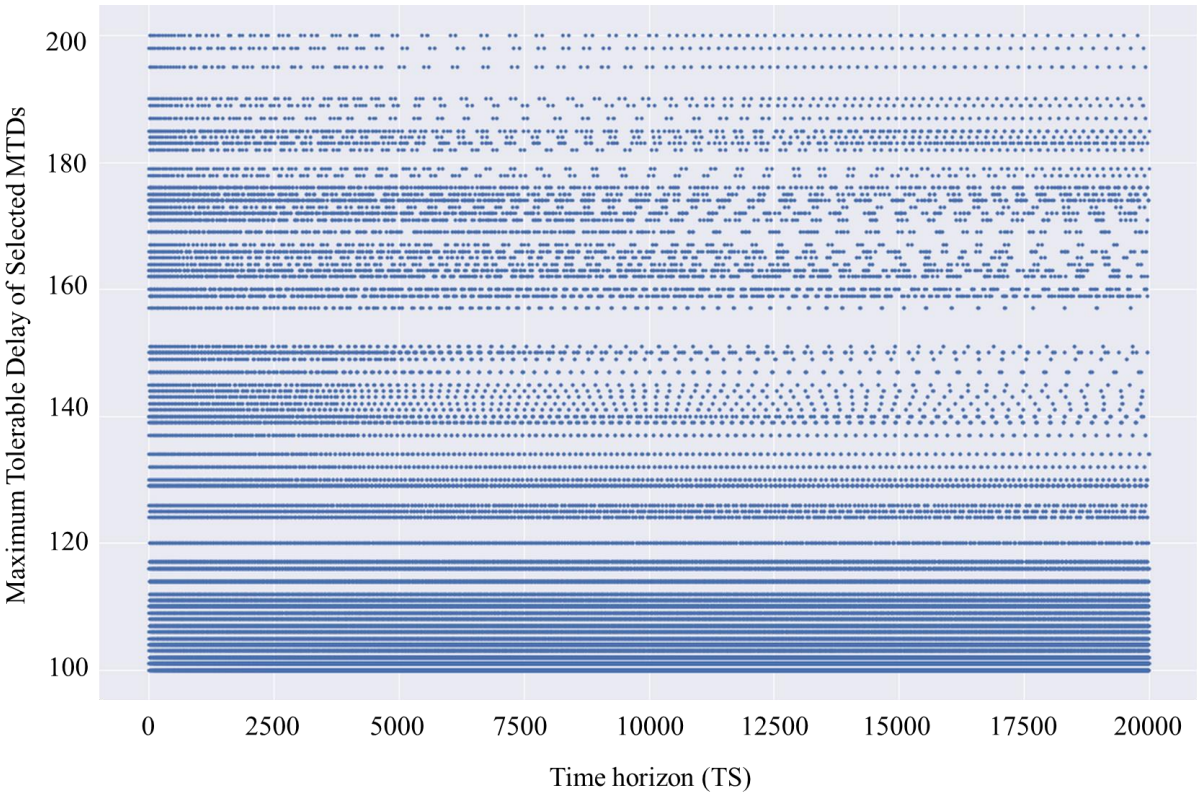


Figure 3.35 Maximum tolerable delay of the selected MTDs by MAB allocations with perfect predictions

3.3.6 Random Allocation Policy

From **[figure 3.36 and figure 3.37],** we can see that a random scheduling algorithm selects the latency completely randomly all the time and the performance of the system is much worse than the proposed sleeping MAB.

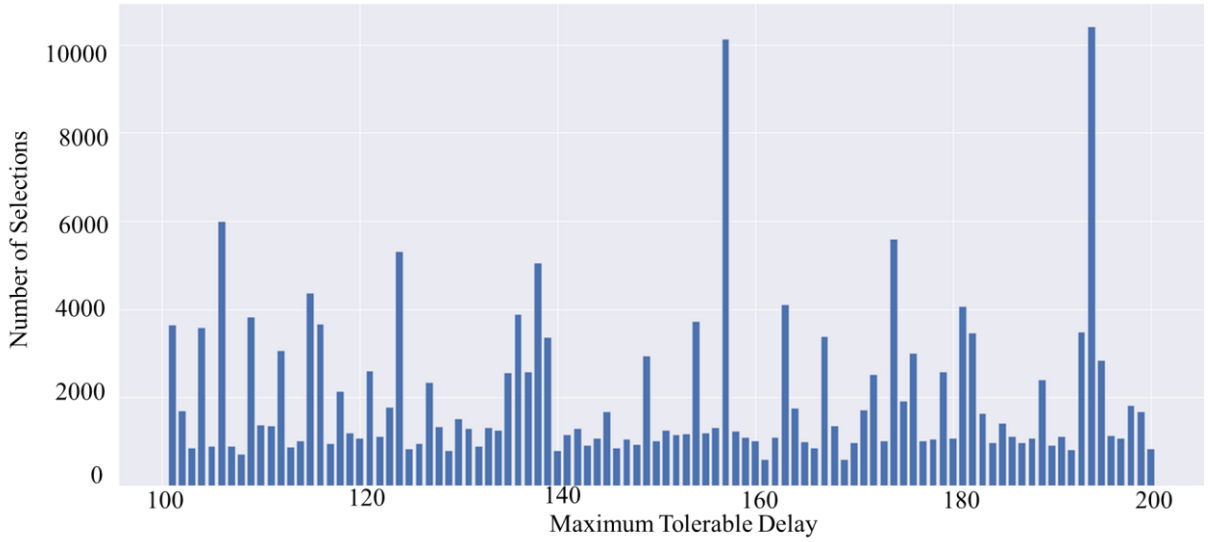


Figure 3.36 Number of times each Maximum Tolerable Delay was selected by random allocation policy

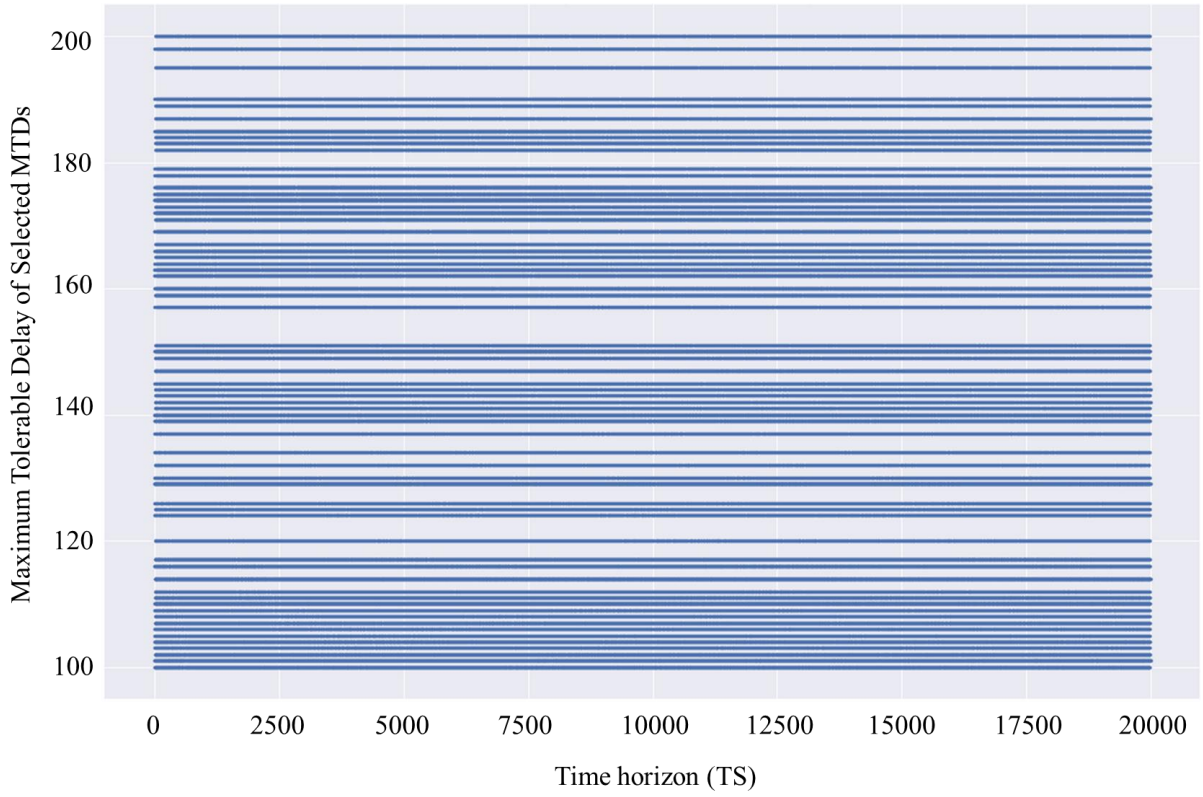


Figure 3.37 Maximum tolerable delay of the selected MTDs by random allocation policy

3.4 Fast Uplink Grant Framework Implementation Summary

[Figure 3.38] shows the summarized graphical representation of the overall proposed Fast Uplink Grant framework implementation discussed thought out this chapter.

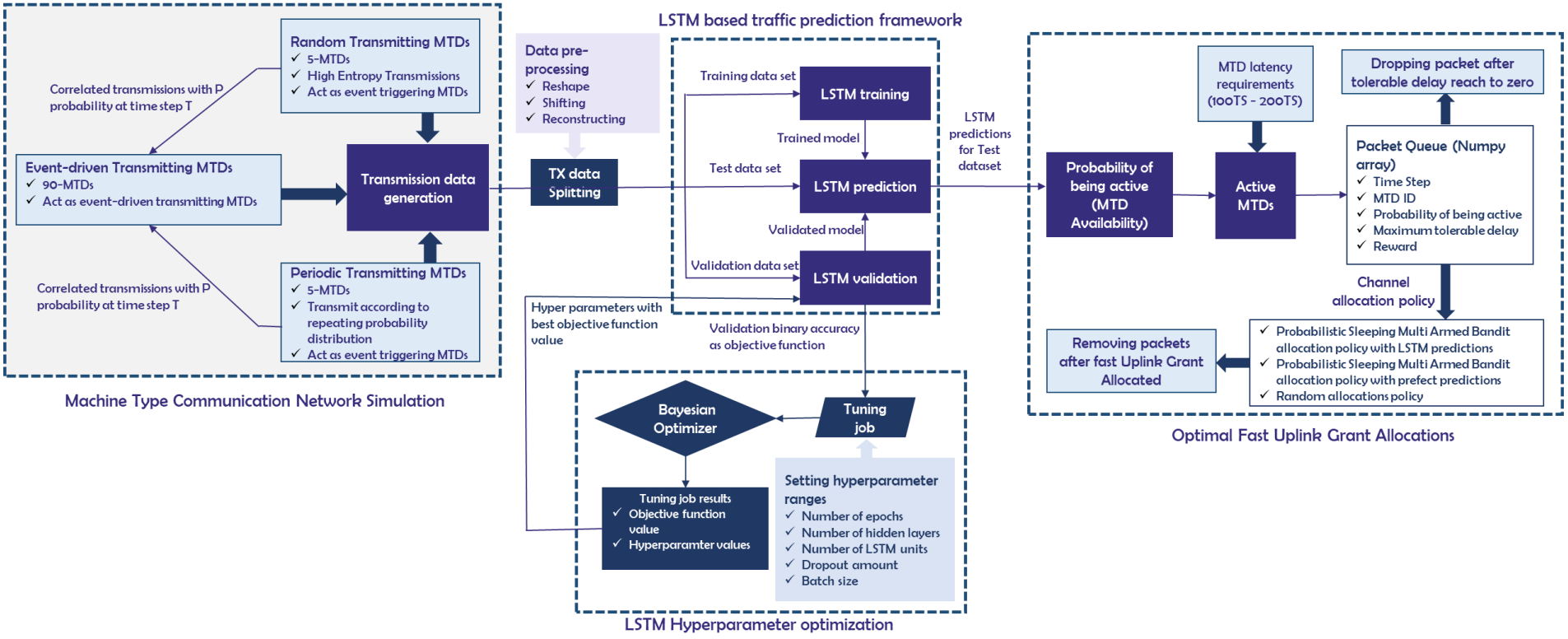


Figure 3.38 Summarized Implementation of overall Fast Uplink Grant Framework

4. RESULTS AND ANALYSIS

4.1 DI Implementation

Directed Information is proposed in ***[DI-Samad]*** as a potential solution for predicting event-driven source traffic prediction. Directed Information values depict the amount of correlation between transmission patterns of considered MTDs in IoT events. However, the performance of the algorithm has not been evaluated in the existing literature. So, this algorithm was implemented to perform a performance evaluation of the results. MATLAB 2019a was used to develop this algorithm.

4.1.1 Data Generation

The first step of implementing the Directed Information was generating data to simulate MTD transmissions. For this purpose, four devices namely X, Y, Z, T were considered. The length of the IoT event was assumed as 12 timesteps. The transmission patterns assumed for the four MTDs are as follows

- For MTD X, a transmission may occur at times t ∈ {1, 2, 3, 4, 7, 8, 9}
- For MTD Y, a transmission may occur at times t ∈ {4, 5, 6, 8, 9, 10, 11}
- For MTD Z, a transmission may occur after 3 timesteps after X has transmitted with a probability of 0.8
- For MTD T, a transmission may occur after 2 timesteps after X has transmitted with a probability of 0.2

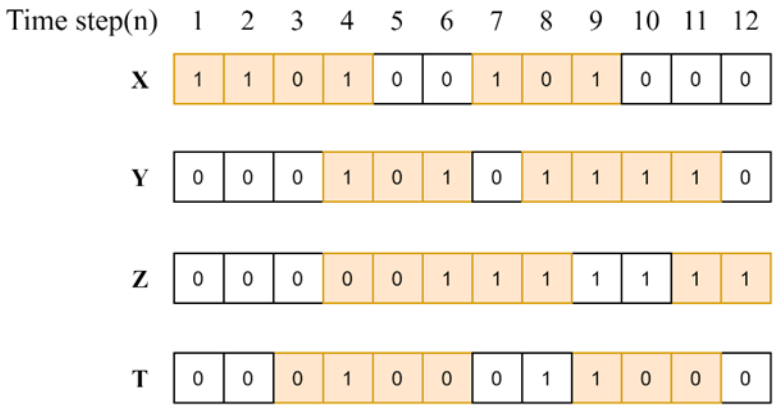


Figure 4.1 Figure name

To create training data and test data sets 100 sequences were generated for each device.

4.1.2 Implementing the algorithm

This algorithm was implemented to compare the transmission sequence of two MTDs as given in the literature. To calculate the Directed Information values the following equation was used.

$$\begin{aligned} I\left(X_k^{k+1} \rightarrow Y_k^{k+1}\right) &= I(X_k; Y_k) + I\left(X_k X_{k+1}; Y_{k+1} Y_K\right) \\ &= H\left(X_k\right) - H\left(X_k Y_k\right) + H\left(X_k X_{k+1} Y_k\right) + H\left(Y_k Y_{k+1}\right) \\ &\quad - H(X_k X_{k+1} Y_k Y_{k+1}) \end{aligned}$$

To calculate the entropy terms on the right-hand side of the equation the following equation

$$H(x_1, \ldots, X_n) = -\sum_{x_1, \ldots x_n} P(X_1, \ldots X_n) \log(P(X_1, \ldots X_n))$$

To carry out the entropy calculations, *entropy Vary* function was coded. To estimate entropy values the joint probabilities for all the combinations of $X_1 \ldots X_N$ is calculated. Consider the following example: to calculate the entropy $H(X_1X_2)$ the following probabilities should be estimated for all the sequences in the dataset as shown in ***[Fig]***

- $P(X_1 = 0, X_2 = 0)$
- $P(X_1 = 0, X_2 = 1)$
- $P(X_1 = 1, X_2 = 0)$
- $P(X_1 = 1, X_2 = 1)$

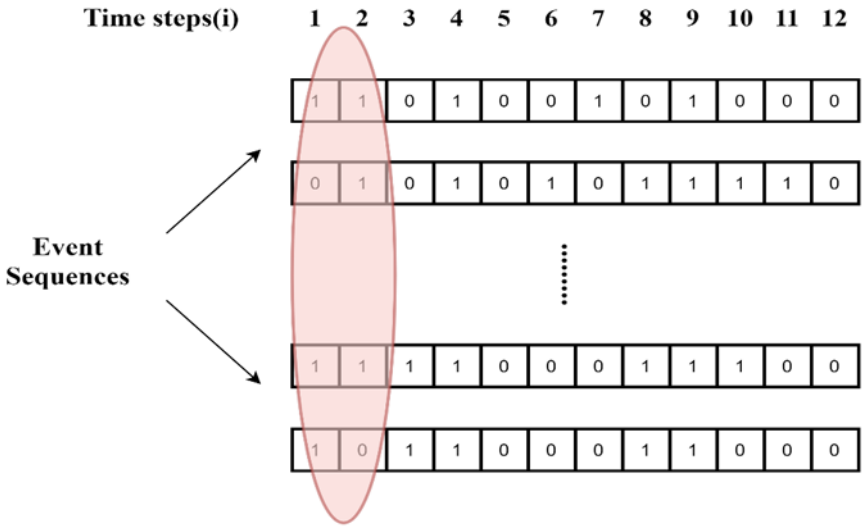


Figure 4.2 Figure name

To calculate the Directed Information value, the *calcDI function* was coded. This function takes the summation of the entropy terms in ***[equation]*** to output the final result which is Directed Information value. The MATLAB code for DI implementation is given in A.

4.1.3 Evaluating the Performance of the Directed Information

Since the performance of the Directed Information algorithm had not been analyzed in the existing literature, a method to measure its performance was formulated. This method will be explained considering the following example: MTD X's transmissions occur in t ∈ {1, 2, 3, 4, 7, 8, 9} and MTD Y's transmissions occurs in t ∈ {4, 5, 6, 8, 9, 10, 11}. The output heat map for Directed Information from X to Y (I (X → Y)) of the DI algorithm is given in [Figure] and the expected heat map for I (X → Y) is given in [Figure].

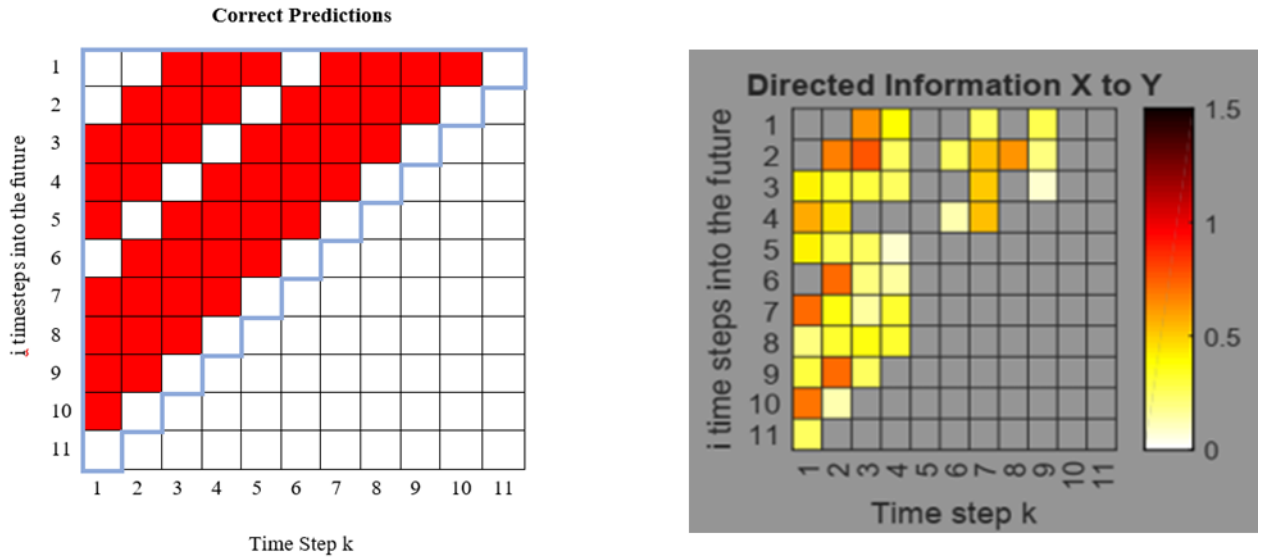


Figure 4.3 Figure name

When X's first timestep is considered, Y's transmissions should occur after 3,4,5,7,8,9,10 the first timestep. Hence the cells (1,3), (1,4), (1,5), (1,7), (1,8), (1,9) and (1,10) are darkened in [Figure]. The same procedure was followed to color cells in [Figure] pertaining to the 2nd to 11th timestep of MTD X. Therefore, the colored cells in [Figure] represent timesteps in which a transmission actually occurs. In [Figure] the colored cells depict the timesteps predicted by the algorithm in which information flow occurs from X to Y. Here the DI information values vary from 0 to 2 and higher the DI values represent a higher chance of transmission occurring in the given timestep.

- If the corresponding cells of the two heat maps are colored, then it was taken as True Positive
- If a cell in the Correct Prediction heat map is colored and its corresponding cell in the Output Heat map is not colored, then it was taken as False Negative
- If a cell in the Correct Prediction heat map is not colored its corresponding cell in the Output Heat map is colored, then we take it False Positive
- If the equivalent cells of the two heat maps are not colored, then take it as True Negative

By using these values, the required metrics such as accuracy, precision, etc. were calculated. The performance evaluation of the Directed Information algorithm is presented in the Results and Analysis section.

4.2 HMM implementation

HMMs are widely used machine learning techniques for solving temporal pattern recognition problems. There are three fundamental types of problems that can be solved using HMMs.

1. Estimate the optimal sequence of the hidden states, when the model parameters are given.
2. Estimate the model likelihood, when model parameters and observed data are given
3. Estimate the model parameters, when only the observed data is given.

Out of the above three cases the event driven traffic prediction scenario fits the thirds case as the sample data to the train the model is available. Therefore, this can be solved using an iterative Expectation-Maximization (EM) algorithm called the Baum -Welch algorithm.

To build the HMM, Python 3.7 was used along with 'hmmlearn' library which has an implementation of the Baum-Welch algorithm. The HMM model was created using Gaussian emissions. To train the model, the dataset used for the Directed Information algorithm was used 80% of transmission

sequences of one MTD was inputted to the model to predict the next 20% of the sequences as illustrated in [Figure].

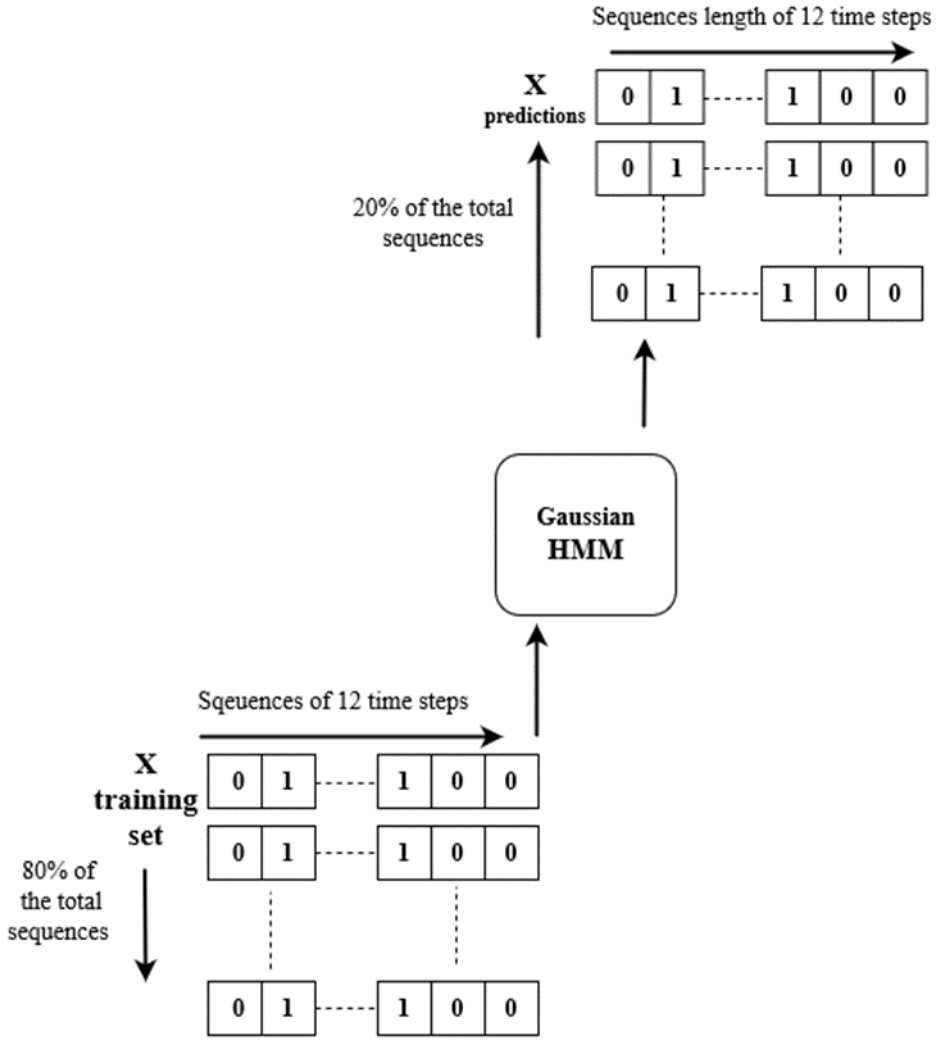


Figure 4.4 Figure name

Due to gradient based optimization in the Baum-Welch, the training can stop at local minima. To avoid this the training process was carried out several times. However, accuracy achieved from this model was around 64%. This due to low accuracy and difficulty in scaling this model to a large IoT network, this model was abandoned.

4.3 Analysis on source traffic prediction

4.3.1 Long Short Term Memory

4.3.2 DI and LSTM comparison

4.4 Analysis on optimal grant allocations

Details about the further analysis done based on the results of implemented fast uplink grant allocation framework presented under this section. Analysis in terms of cumulative reward, cumulative regret over time and analysis in terms of latency requirements has presented in this section. Especially, effect

of the accuracy of the source traffic predictions framework on allocation has been investigate under this analysis.

4.4.1 Reward Analysis of different allocation policies

From bottom to top **[Figure 4.5]** shows the result is compared to: a) A random allocation policy, b) proposed probabilistic sleeping MAB allocation policy c) A scenario in which the prediction is error free with the proposed probabilistic sleeping MAB allocation policy d) imaginary rewards where the maximum possible reward consider for each time step. **[Figure 4.5]** clearly shows that the random allocation of radio resources has linear lowest cumulative reward over time which is much worse compared to the positive exponential reward achieved by the proposed solution. Moreover, **[Figure 4.5]** shows that perfect prediction has the best performance. The baseline random policy performs very poorly in terms of reward as seen from Fig. 3 since its reward increases linearly with time.

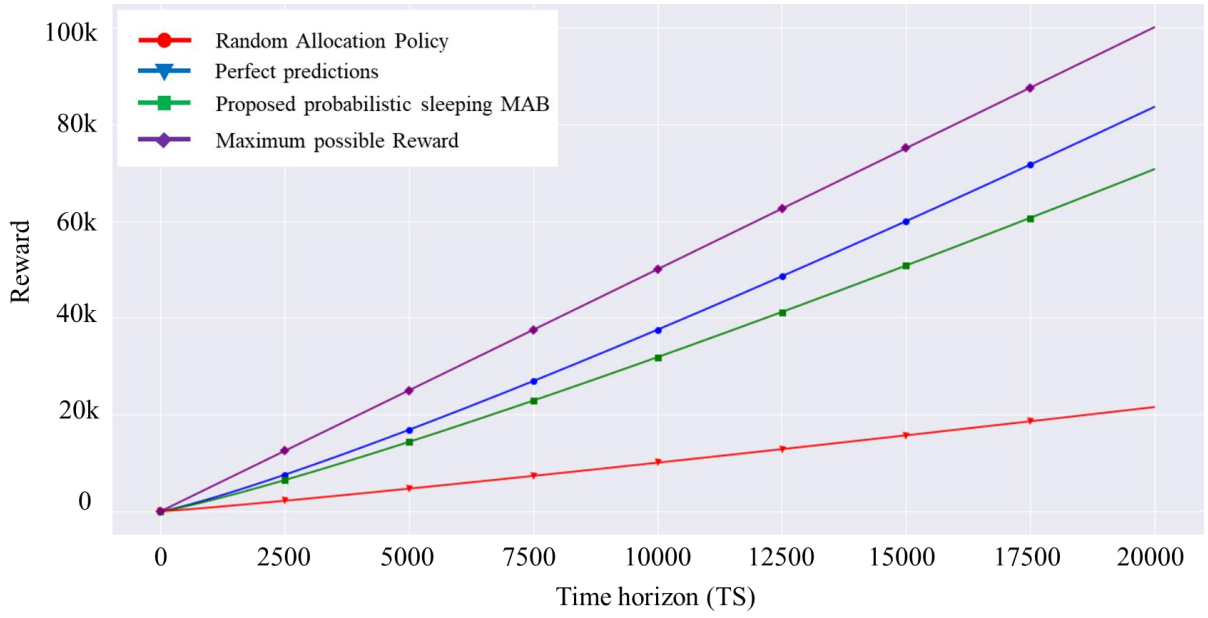


Figure 4.5 Reward resulting from the proposed probabilistic MAB compared to probabilistic MAB with perfect prediction, and random allocation

From bottom to top **[Figure 4.6]** shows the resulting regret is compared to: a) A scenario in which the prediction is error free with the proposed probabilistic sleeping MAB allocation policy, b) proposed probabilistic sleeping MAB allocation policy c) A random allocation policy. **[Figure 4.6]** clearly shows that the random allocation of radio resources has linear highest cumulative regret over time which is much worse compared to the logarithmic regret achieved by the proposed solution. Moreover, **[Figure 4.6]** shows that perfect prediction has the best performance. The baseline random policy performs very poorly in terms of regret as seen from Fig. 3 since its regret increases linearly with time.

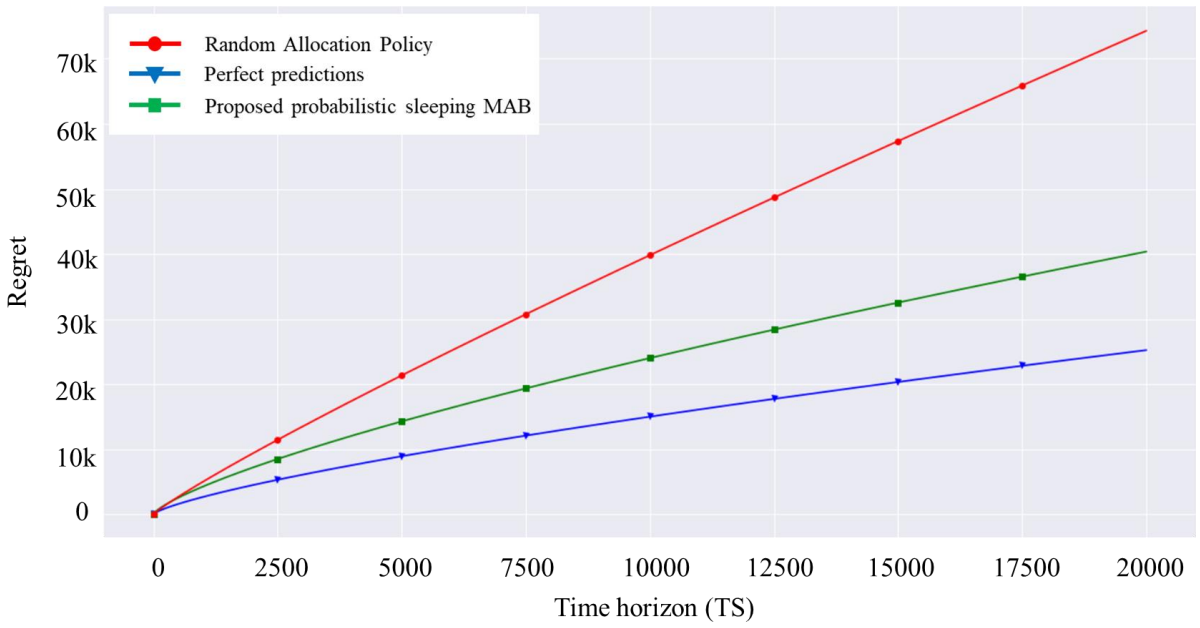


Figure 4.6 Regret resulting from the proposed probabilistic MAB compared to probabilistic MAB with perfect prediction, and random allocation

4.4.2 Latency Analysis of different allocation policies

In **[Figure 4.6]**, from bottom to top we present the average delay of the selected MTDs for a) A scenario in which the prediction is error free with the proposed probabilistic sleeping MAB allocation policy, b) proposed probabilistic sleeping MAB allocation policy c) A random allocation policy. In simple terms this average delay of the selected MTDs represent the average amount of timesteps that MTD has to wait unlit receive Fast Uplink Grant for packet transmissions.

The proposed probabilistic sleeping MAB algorithm can provide much better average achieved access delay in the system. One must note that the achieved average access delay is almost constant for any value of the maximum tolerable delay, since the select MTDs are averaged. This shows that, in real-time systems, by increasing the number of MTDs, our proposed solution achieves almost a three-fold improved performance compared to baseline methods. This is an interesting result since it shows that, for a massive access scenario, our proposed method can achieve very low access delay. In contrast, conventional random access based systems experience excessive delays due to collisions.

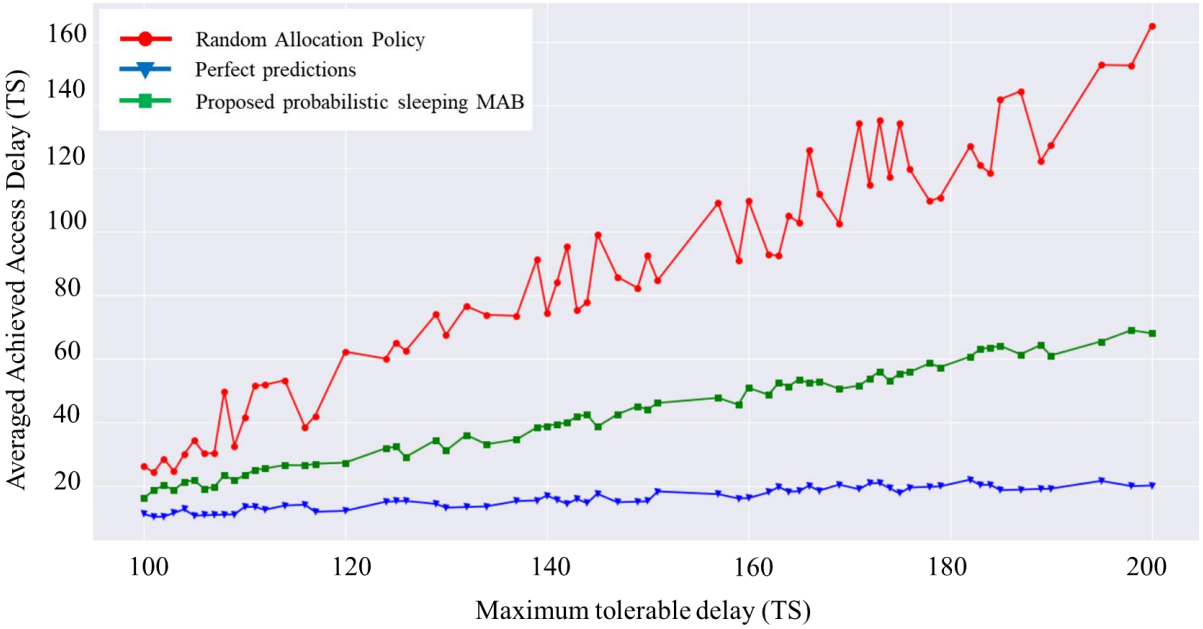


Figure 4.7 Average achieve access delay of MTD selections

By considering **[figure 4.4]** to **[figure 4.7]** it is clear that there is always difference in terms of reward, regret and achieved average latency when LSTM predictions and perfected perfections use as the MTD availability for proposed probabilistic MAB based allocation framework.

5. SUMMARY AND CONCLUSION

5.1 Summary

5.2 Conclusion

5.3 Recommendations