A Machine Learning based Prediction Model for Bursty Traffic in 5G mMTC Networks

AASMUND SØRAA

SUPERVISORS
Professor Frank Yong Li
Dr. Indika A. M. Balapuwaduge
Ph.D. candidate Thilina Nuwan Weerasinghe

University of Agder, 2019
Faculty of Engineering and Science
Department of Information and Communication Technologies

Abstract

The rapid growth in popularity for the Internet of Things (IoT) has paved the way for a new type of communication, called Machine-type Communication (MTC), causing substantially increased traffic volume. The existing network infrastructure is facing challenges when handling the increased traffic load, especially under bursty conditions. Consequently, the network will more often experience congestion during bursty conditions, which will in turn reduce throughput, and lead to a decline in quality of service.

Several improvements have been studied to help support the increasing mMTC traffic load. However, most of these improvements are reactive, meaning that they are only enabled when congestion is detected at the eNB. By predicting the occurrence of congestion in advance, we can proactively implement the prediction based improvements and conceivably avoid congestion all together.

In this thesis, we propose a machine learning (ML) based framework for forecasting traffic arrival, and subsequently predict the occurrence of congestion in bursty traffic conditions. The framework is composed of two ML algorithms, each responsible for a specific task. The first algorithm is based on a long short-term memory (LSTM) architecture, a common type of recurrent neural network (RNN) suitable for modeling long-term time-series data. The second algorithm is based on a neural network used to classify whether or not the traffic is leading to congestion. We test our solutions in various scenarios with additional sub-cases under different network conditions. In our tests we look at how the information available at the eNB affects traffic forecasting, and how this in turn affects the congestion prediction.

Our proposed framework achieves precise results in forecasting traffic arrival for certain traffic conditions. We observe that for predicting congestion in the bursty window, the already available data performs equally well or better than the forecasted data in all cases. With the original detected traffic, we achieve 100% accuracy on predicting congestion using the proposed classification model. By implementing ML techniques at the eNB, we can reduce congestion and improve throughput in the network.

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science in Information and Communication Technology at the University of Agder. This thesis is based on work done in conjunction with the IKT590 course, which is a 30 ECTS credit course in the Information and Communication Technology Master's Programme at the University of Agder, Norway. This work was started on February 4th 2019, and submitted on May 24th 2019.

I would like to thank my supervisors: Professor Frank Y. Li; Dr. Indika A. M. Balapuwaduge; and Ph.D candidate Thilina N. Weerasinghe, for their guidance and help throughout this project period. Their knowledge and expertise on the topics for this thesis have been invaluable.

Aasmund Søraa
Grimstad
May 24th, 2019

Contents

Contents **iii**

List of Figures **v**

List of Tables **vii**

1 Introduction **1**
- 1.1 Background and Motivation . . . 1
- 1.2 Problem Statement . . . 3
- 1.3 Research Approach . . . 3
- 1.4 Thesis Outline . . . 4

2 Enabling Technologies and Related Work **5**
- 2.1 LTE/LTE-A/5G New Radio . . . 5
- 2.2 MTC and Massive MTC . . . 6
- 2.3 Random Access in LTE/LTE-A/5G-NR . . . 7
- 2.4 Duty Cycling and Wake-up Radio . . . 9
- 2.5 Machine Learning . . . 9
  - 2.5.1 Neural Networks and Deep Learning . . . 10
  - 2.5.2 Recurrent Neural Network . . . 11
  - 2.5.3 Long Short-Term Memory . . . 12
- 2.6 Related Work . . . 13

3 Proposed Solution **15**
- 3.1 Network Scenario and Assumptions . . . 15
- 3.2 Proposed Solutions . . . 18
  - 3.2.1 Future Prediction Model . . . 19
  - 3.2.2 Congestion Classification . . . 21
- 3.3 Traffic Pattern Analysis . . . 21

4 Algorithm Implementation and Model Evaluation **25**
- 4.1 Algorithm Overview . . . 25

4.2 Test Scenario Overview 27
4.3 Results for Test Scenario 1 28
4.3.1 Test Results for Case 1A 29
4.3.2 Test Results for Case 1B 31
4.4 Results for Test Scenario 2 36
4.5 Results for Test Scenario 3 38
4.6 Summary of Previous Chapters and Results 40

5 Factors Relevant to Numerical Results and Further Discussions 41
5.1 Data Normalization and Standardization 41
5.2 Machine Learning Parameters 41
5.2.1 Activation Functions 42
5.2.2 Network Complexity 42
5.3 Multi-step Forecasting and Loss Calculation 43
5.4 Addition of Normal Traffic 43
5.5 Number of Samples in Dataset 44
5.6 State Space 44
5.7 Further Discussions 44
5.7.1 Processing Time for Traffic Prediction 44
5.7.2 A Potential Algorithm for Preamble Allocation at the eNB 45

6 Conclusions and Future Work 47
6.1 Conclusions 47
6.2 Thesis Contributions 48
6.3 Future Work 49

Bibliography 50

Appendices 54

A Additional Simulation Results 55
A.1 Effect of Using Various Combinations of Complexity on the Traffic Prediction Model 55
A.2 Effect of Additional Epochs During Training 62

B Python/Pytorch Code Samples 63
B.1 Algorithm 4.1 63
B.2 Algorithm 4.2 67
B.3 Algorithm 4.3 69

List of Figures

2.1 Contention based RACH process. 7
2.2 Branches of machine learning. 10
2.3 Feed-forward neural network. 11
2.4 Recurrent neural network. 12

3.1 Probability of at least 1 collision in a given RA slot using 54 preambles. 17
3.2 Overview of solution framework. 19
3.3 A two layer LSTM used for multi-step prediction by predicting one step at a time. 20
3.4 Bursty traffic patterns for 10000 active devices. 21
3.5 Bursty traffic patterns for 20000 active devices. 22
3.6 Bursty traffic patterns for 30000 active devices. 22
3.7 Comparison of RA attempts. 23
3.8 Successful attempts per slot for 10 different simulations of 10k, 20k, and 30k active devices. 24

4.1 Model of the traffic prediction network. 28
4.2 Multi-step prediction using 100 RA slots as input seed. 30
4.3 Predictions from HG 1 visualized with different length input seeds. 33
4.4 Predictions from HG 2 visualized with different length input seeds. 34
4.5 Predictions from HG 3 visualized with different length input seeds. 35
4.6 Classification model. 37

5.1 Proposed implementation of framework at eNB. 45

A.1 Effect of Different Number of Hidden Nodes in 2-layer LSTM for Forecasting Traffic based on 10k Active Devices. 56
A.2 Effect of Different Number of Hidden Nodes in 3-layer LSTM for Forecasting Traffic based on 10k Active Devices. 57
A.3 Effect of Different Number of Hidden Nodes in 2-layer LSTM for Forecasting Traffic based on 20k Active Devices. 58
A.4 Effect of Different Number of Hidden Nodes in 3-layer LSTM for Forecasting Traffic based on 20k Active Devices. 59

A.5 Effect of Different Number of Hidden Nodes in 2-layer LSTM for Forecasting Traffic based on 30k Active Devices. 60
A.6 Effect of Different Number of Hidden Nodes in 3-layer LSTM for Forecasting Traffic based on 30k Active Devices. 61
A.7 Effect of Additional Training on Traffic Forecast for HG3. 62

List of Tables

3.1 Bursty traffic performance (LTE FDD) as mentioned in [15] 16
3.2 Simulation parameters for LTE RACH 17
3.3 Ratio of detected attempts vs arrivals during RA 23

4.1 Overview of scenarios and cases 28
4.2 Common model parameters and functions used in all experiments 29
4.3 Effect of using different input lengths measured in MSE between real and predicted traffic 30
4.4 HG 1 with MSE 32
4.5 HG 2 with MSE 32
4.6 HG 3 with MSE 33
4.7 HG 1 accuracy 37
4.8 HG 2 accuracy 37
4.9 HG 3 accuracy 38
4.10 Accuracy on Scenario 3 39

List of Algorithms

4.1 Data generation 26
4.2 Traffic prediction 26
4.3 Congestion prediction 26

Acronyms

3GPP 3rd Generation Partnership Project.

ACB Access Class Barring.

AI Artificial Intelligence.

AN Access Network.

BS Base Station.

DL Deep Learning.

eNB Evolved NodeB.

HG Heterogeneous Group.

HTC Human-type Communication.

IoT Internet of Things.

ISM Industrial, Scientific and Medical.

ITU International Telecommunication Union.

LSTM Long Short-term Memory.

LTE Long-Term Evolution.

M2M Machine-to-Machine.

ML Machine Learning.

mMTC massive Machine-type Communication.

MSE Mean Squared Error.

MTC Machine-type Communication.

NN Neural Network.

NOMA Non-orthogonal Multiple Access.

NR New Radio.

NSA Non-Standalone.

OFDMA Orthogonal Frequency Division Mtultiple access.

PRACH Physical Random Access Channel.

RA Random Access.

RACH Random Access Channel.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

SA Standalone.

SIB System Information Block.

UE User Equipment.

WSN Wireless Sensor Network.

WuR Wake-up Radio.

Chapter 1

Introduction

With the recent paradigm shift in information gathering, the way we interact with other people and businesses has drastically changed. Almost any interaction is now feasible using digital communication, and the added convenience of this ensures the trend of digitalization will continue. This new paradigm shift has been instrumental in forming the smart city concept, which along with the various other smart *X*[1] concepts has had a tremendous growth over the last decade. The main pillar of smart *X* is the massive amount of data available, and the possibility of communicating this data to other devices via the Internet of Things (IoT). This new principle of automatically transmitting and collecting data causes a far more congested network which creates challenges not fully considered previously. For cellular based communication, a lot of effort is therefore put into improving network quality in the coming 5G standard, and ensuring that the technology is suitable for the massive growth in smart devices [1]. In this chapter, we provide background information and introduce the problem statement for this thesis. A description of the research approach follows, as well as a section on the layout of the thesis.

1.1 Background and Motivation

IoT is a concept where ordinary devices and services are fitted with technology, enabling them to connect to the Internet and communicate with other devices and services.

Most communication technologies can be used for IoT. Normally, wireless Internet communication standards utilizing the Industrial, Scientific and Medical (ISM) bands have been used for IoT traffic. However, with advancements in the technology for telecommunications, Long-Term Evolution (LTE) and its successors are also used. 3rd Generation Partnership Project (3GPP), which are responsible for cellular communication standards, include advancements

[1]Smart *X* is a terminology in this thesis which refers to smart “everything”, such as: smart city, smart health, smart vehicles, smart gadgets, smart house, etc.

for IoT traffic in their LTE/LTE-A/5G standards, but under the nomenclature Machine-type Communication (MTC).

With the expected growth of MTC in the near future, the existing LTE technology created for mainly Human-type Communication (HTC) will have congestion problems, leading to issues for both humans and machines. For LTE based communication, one of the largest bottlenecks is the Random Access (RA) procedure initialized every time a device wishes to communicate with the Base Station (BS), also known as Evolved NodeB (eNB). The RA procedure is performed between the device and the eNB in the same manner for both MTC and HTC. The eNB can receive numerous requests simultaneously, and rely on a pre-determined set of instructions to ensure that the different requests do not collide during the RA procedure. In normal scenarios the transmission interval for MTC devices is such that the current RA scheme is sufficient to process all attempted transmissions. Under abnormal scenarios, also called *bursty traffic*, the access success probability drops drastically as soon as the number of devices increases. Bursty traffic occurs when a large number of devices attempt to transmit data, such as after a power failure or similar event, simultaneously. A major cause of the congestion is the re-transmissions following collisions and the subsequent cascading failures. With so many different types of communication carried out using cellular communication, it is non-trivial to find a solution that is optimal for all scenarios.

The Orthogonal Frequency Division Mtultiple access (OFDMA) scheme is used in LTE to ensure multiple devices are able to connect an eNB during the same access slot. With OFDMA, the eNB is able to distinguish 64 different devices at each slot as a result of the orthogonal attribute of OFDMA. But, in scenarios with rapidly increasing arrivals, the number of devices attempting to connect might be so large that a significant congestion problem still occurs. The current RA procedure involves each device randomly selecting one of the available preambles in a given RA opportunity, where the number of preambles is at most 64. In the case of two or more devices selecting the same preamble, a collision will occur. In some cases of collision the eNB is still able to distinguish the preamble sent, but as the number of devices selecting the same preamble increases, the probability of the eNB correctly decoding the signal decreases.

To avoid collision at the eNB, certain improvements which reduce the number of attempts per slot have been studied. However, most of the improvements are reactive and action are not taken until after congestion has occurred. By predicting if the traffic will lead to congestion or not, we can proactively implement traffic improvements to reduce or avoid congestion. Little work has been done for predicting congestion in a bursty mMTC scenario based on traffic detected at the eNB. This triggers our motivation to analyse how traffic arrivals can help in predicting congestion, using both forecasted and real traffic. We introduce two machine learning based models to predict the attempts and subsequent congestion in bursty mMTC scenarios.

1.2 Problem Statement

The access procedure in LTE/LTE-A/5G-NR is based on slotted access. Whenever a device has new data to send on the uplink, it has to wait on the next access opportunity to connect to the eNB before proceeding with the transmission. As the number of devices with data to transmit increases, the number of devices transmitting in the same access opportunity also increases. This in turn leads to a drastic increase in collision probability.

We refer to the number of new devices ready to send data on the uplink as *arrivals*. *Attempts* are the number of new arrivals plus the re-transmissions caused by collisions. *Detected attempts* are attempts that have successfully been detected by the eNB. We assume the eNB is unable to detect the signal when two or more devices choose the same preamble in the same access opportunity, and all the devices experiencing collision will then have to re-transmit. Environmental factors, such as signal-to-noise ratio, are not considered in this thesis. With only 64 preambles per cell, the collision probability becomes significant as the number of attempts per RA opportunity increases [8].

The main objective of this thesis is to predict the detected attempts in future RA slots. We presume that having information about future traffic will allow the eNB to be proactive in deploying counter-measures, and in turn avoid congestion before the traffic reaches the critical point. Some counter-measures are more effective in certain conditions, such as low traffic volume, while others are more effective in conditions with large volume. By forecasting traffic, we can implement effective counter-measure based on the specific traffic condition. The main objective will be evaluated using Mean Squared Error (MSE) at different time steps. The goal is to predict as many time steps ahead as possible.

The secondary objective is to classify whether or not the network traffic is leading to congestion. Although most congestion stem from bursty traffic, not all bursty traffic lead to congestion. Congestion only occurs when the number of arrivals is above a certain threshold, and therefore implementing actions to prevent congestion might not be necessary in every case of bursty traffic. The secondary goal is evaluated based on classification accuracy.

1.3 Research Approach

The literature review started with researching previous work on the topic of traffic prediction in mobile network. Some work was found during the review which studies the topic in various fashions [3][4][6], but they look either at only event-driven traffic, or a single number of active devices. Subsequent research was done on the existing improvements for LTE as proposed by 3GPP. A lot of research has been done by other parties in the proposed improvement areas.

Methods to improve the RA process are still being studied, with much of the focus now on the topic of Non-orthogonal Multiple Access (NOMA), which is expected to be the access mechanism in 5G phase 2, as opposed to OFDMA.

Afterwards, both theoretical and practical data collection scenarios were studied. For the theoretical part a lot of research has been done by 3GPP which is still relevant. Data collected in testbeds is lacking, but relevant trace data was found for a smaller IoT network [5]. Because of the scarce data availability on relevant real-life data, this thesis is based on simulated data according to models by 3GPP [15].

Machine learning methods were studied to find a suitable architecture for the available data. Previous work has been done using decision tree algorithms [4], and for novelty this thesis focuses on different methods for time-series prediction. Various ways of modeling the data were considered for best result. Machine learning algorithms require substantial trial-and-error with regards to parameter tweaking, and experiments were conducted to find the best parameters for the data.

1.4 Thesis Outline

The rest of the thesis is organized as follows:

- Chapter 2 gives an introduction to the relevant technologies considered in this thesis, as well as further details on related works.
- Chapter 3 presents a general overview of the proposed solution and delves further into the assumptions considered in this thesis. Analysis of the data considered is also presented.
- Chapter 4 shows the results and experiments performed. The results are then discussed, and a review of the previous chapters in relation to the results is given.
- Chapter 5 discusses the factors responsible for the numerical results, and details the effect of using certain features over others.
- Chapter 6 concludes this thesis and lists main contributions. The chapter also outlines topics for future research.

Chapter 2

Enabling Technologies and Related Work

The future of telecommunications is evolving more and more towards the concept of smart *X* and massive Machine-type Communication (mMTC). Moving towards smaller cells and denser deployment allows the massive number of devices to select cells with best transmission opportunity. A simple solution to the congestion problem would be to just deploy more base stations, and although new technological advances such as self organizing networks makes this approach more and more feasible, it is often better to improve the technology behind the network in order to enhance scalability.

This chapter will present some of the underlying technologies used for communications and machine learning in relation to this work. Then a review of some selected related work is presented.

2.1 LTE/LTE-A/5G New Radio

LTE is a standard for wireless mobile communication developed by 3GPP. It is a continuation of the 3G mobile network with focus on higher data rates and better quality of service [11]. The packets in LTE are based purely on IP, while the access solution is based on OFDMA. This allows for high data rates for multiple users at a time. The Access Network (AN) in LTE is a network of eNBs inter-connected with each other. The eNB will broadcast system information to User Equipment (UE) and devices in the area with information on how to connect to the eNB and which settings to use for connecting. There are a couple of different types of eNB differentiated by the coverage size of each station. A macro-cell is the largest, followed by micro-, pico-, and femto-cell. A cell, such as a femto-cell residing inside a building, can exist within a larger cell [14].

LTE-Advanced is a standard by 3GPP developed to further improve on LTE and to fulfill the initial goals of the proposed 4G standard. LTE-A was standardized in 3GPP release 10 and fulfilled the requirements for 4G by notably enabling gigabit connections by introducing Carrier Aggregation, enhanced MIMO and support for Relay Nodes etc. [12] [13].

5G New Radio (NR) is a term used to describe the newest and ongoing releases from 3GPP. It enhances LTE-A towards a new generation of mobile communication standards. 5G phase 1 is standardized by 3GPP in Release 15, with the first version of Release 15 released in March 2018. 5G NR will be deployed with two architectures: Standalone (SA) and Non-Standalone (NSA) architecture. The NSA architecture is intended for deployment with already existing LTE infrastructure, while the SA architecture is intended for new areas without existing infrastructure. The term "New Radio" refers to the AN introduced in the 5G standard. Differences between the AN in 5G and LTE is the new eNB, where the eNB in NR are referred to as: and gNB, in SA; en-gNB, in NSA architecture. In NSA, the en-gNB and eNB stations will co-exist and are connected to the same network [16]. Some differences exists between the eNB for LTE and gNB for 5G NR, but the number of preambles available for Random Access Channel (RACH) remains 64 in the new generation, as 5G NR is also OFDMA based [17]. NOMA schemes, as opposed to OFDMA, have been proposed to 3GPP as solutions for the mMTC congestion problem. NOMA has been included as a study item in 3GPP release 13 and as a work item in release 14, and is a candidate for 5G phase 2 in release 16 [23].

2.2 MTC and Massive MTC

MTC, also known as Machine-to-Machine (M2M) communication, is the communication between devices; either directly between the devices or via a centralized MTC server or servers. MTC has been standardized by 3GPP in Release 9 and their LTE-M standard is developed for low-power, wide-area communication between devices to compete with other technologies and communication standards for the Massive IoT market. NB-IoT is another standard developed by 3GPP which also focuses on low-power, wide-area communication for MTC devices. The main difference between NB-IoT and LTE-M is NB-IoT having a bandwidth of 180KHz, and LTE-M a bandwidth of several MHz. The benefit of using NB-IoT is that it has more flexible deployment possibilities compared to LTE-M, which is restricted to the LTE band, as well as lower cost [19].

With the increased complexity and affordability of MTC devices, utilization of MTC technology is growing quickly, and it is estimated that there will be over 1 billion connected cellular devices in 2020 [18]. This massive increase in MTC traffic will demand more specific technology developed with this type of communication in mind. To establish a precise understanding of the scale of mMTC, International Telecommunication Union (ITU) requires support for 1 million devices per square kilometer in 5G New Radio [10], and Nokia expects to be able to serve 1 million devices per cell using their proprietary technology [18]. Another paper on mMTC [20] highlights standard assumptions of mMTC as:

- Small, byte-sized packets.
- Large number of users per cell, up to 300,000.
- Dominated by infrequent up-link transmissions.
- Low data rate.
- Mix of event-driven and periodic traffic.
- Battery-constrained devices.

Looking at the mMTC definitions and requirements from [20] and [21], we observe that most services utilizing MTC network are services such as smart-meters, monitoring- and tracking systems, payment systems (vending machines etc.), sensor systems for access control and security, and more. Most, if not all of these services, fulfill the requirements and assumptions listed above. Many of these services, such as security sensors and monitoring systems, also require low-latency which can be heavily impacted in a mMTC network during bursty conditions.

2.3 Random Access in LTE/LTE-A/5G-NR

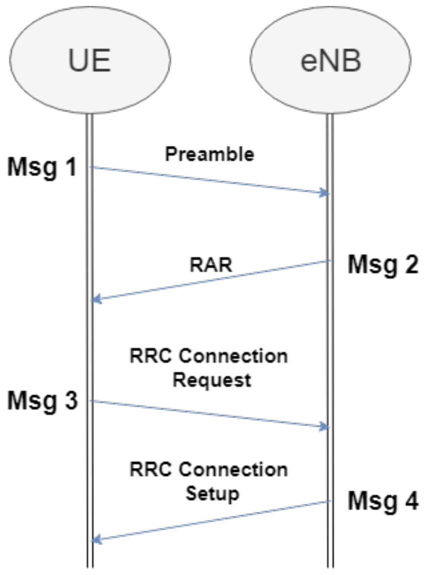


Figure 2.1: Contention based RACH process.

RACH is part of the initial connection procedure performed to synchronize the UE or device with the eNB in cases such as: power on, data upload, data download, and during handover. The eNB uses a set of preambles to uniquely identify devices or UEs during RACH. The preambles are included in the eNB's System Information Block (SIB), which it broadcasts periodically. The preambles are Zadoff-Chu [22] sequences, complex mathematical sequences which has the property that shifting the sequence cyclically creates a new sequence with zero correlation to other shifted sequences. This ensures that each device with a distinct preamble is distinguishable from the other devices in the cell. The devices also receives timing synchronization and other settings related to random access via the SIB. The device, or UE, will select a random preamble from the set and use this to setup a connection with the eNB and facilitate further connection requests.

There are two possible scenarios during RACH: contention-based, and contention-free RACH. In contention-based RACH, the devices will randomly select a preamble from the same pool, and there is a risk that two or more UEs select the same preamble, which will cause a collision. For contention-free RACH, the device will *receive* a preamble from the eNB, which ensures that the device does not experience collision. Contention-free RACH is used for example in the case of handover, and the preambles used are a subset of the total preambles generated for each cell. The number of preambles designated for contention-free RACH is included in SIB and it is possible to change this value according to qualitative preference.

The transmission of a preamble from the device to the eNB is called "MSG 1". If the eNB is able to detect the signal, it replies with a Random Access Reply (RAR) message, "MSG 2", with a specific Radio Access Radio Network Temporary Identifier (RA-RNTI) value calculated based on information from the preamble which identifies the sending device. The message also includes a Temporary Cell Radio Network Temporary Identifier (Temporary C-RNTI) for use in later messages. In the case that multiple devices send the same preamble they would all receive the same RAR. A RAR timeout value is included in the SIB and informs the device of how many sub-frames it can expect to wait before the RAR arrives. If the RAR does not arrive within this window, the device will retry MSG 1, possibly with a larger power value and after a specific backoff period if indicated in the SIB.
"MSG 3" contains the L2/L3 messages, such as RRC Connection request, and is sent on the physical up-link channel (PUSCH). The devices will also send a generated unique ID to the eNB for contention resolution. The eNB will reply in "MSG 4" with a Hybrid Automatic Repeat Request (HARQ) to the device, or devices in the case of multiple using the same preamble and the Temporary C-RNTI from MSG 2.
The eNB will send a contention resolution message addressed to the Temporary C-RNTI with the unique ID of the device. In the case of a collision the other devices will register that the unique ID does not match theirs, and they will stop the connection phase and retry MSG 1. The device with the correct ID will send an acknowledgment to the eNB and start transmitting data on the uplink.
For contention-free RACH, the process does not include the contention resolution phase, and

will go to the data uplink phase after RAR is sent from the eNB [24].

2.4 Duty Cycling and Wake-up Radio

In many wireless sensor networks (WSNs), the sensors are left unattended after deployment. This means that battery recharge or replacement is not performed and it is important that sensors are able to conserve energy whenever possible.

Duty cycling is a traditional technique used to conserve energy in WSNs where the sensors are put to sleep in periods where they do not have any data to transmit. However, duty cycling requires sensors to listen for data after each period, which can be costly in the event that the sensor receives infrequent data [25]. A modern technique for energy conservation is the use of Wake-up Radio (WuR) instead of duty cycling [26]. WuR operates on an on-event basis which reduces energy spent on waking up after each period. WuR combines the sensor with a separate receiver responsible for waking up the sensor on-event, and ensures energy consumption is minimized while idling [27] [28].

With the move from traditional duty-cycling techniques with deterministic transmission intervals, to WuR with stochastic transmission intervals, the amount of event-triggered mMTC traffic, which is a major instigator for bursty traffic conditions, increases.

2.5 Machine Learning

Machine Learning (ML) is a discipline in computing where the solution is not explicitly programmed. Computers instead learn the solution, or an approximation thereof, based on trail and error. It is a subset of Artificial Intelligence (AI) and is a well researched field with massive growth in popularity following recent advances in computational power and data gathering. ML uses datasets to train computers to recognize patterns and solve tasks, and uses various algorithms to achieve this. We can categorize the various algorithms into three main groups: supervised-, unsupervised-, and Reinforcement Learning (RL) algorithms [29].

- Supervised machine learning algorithms use a labeled dataset to train the computer based on truthful inputs and outputs. This approach ensures that the computer will know what the expected output should be, and the goal is to train the network such that it learns an approximate function, and is able to generalize to new, unseen data. Supervised learning can be split into *classification* tasks and *regression* tasks. Classification tasks are tasks with a discrete state space, while regression tasks have a continuous state space.

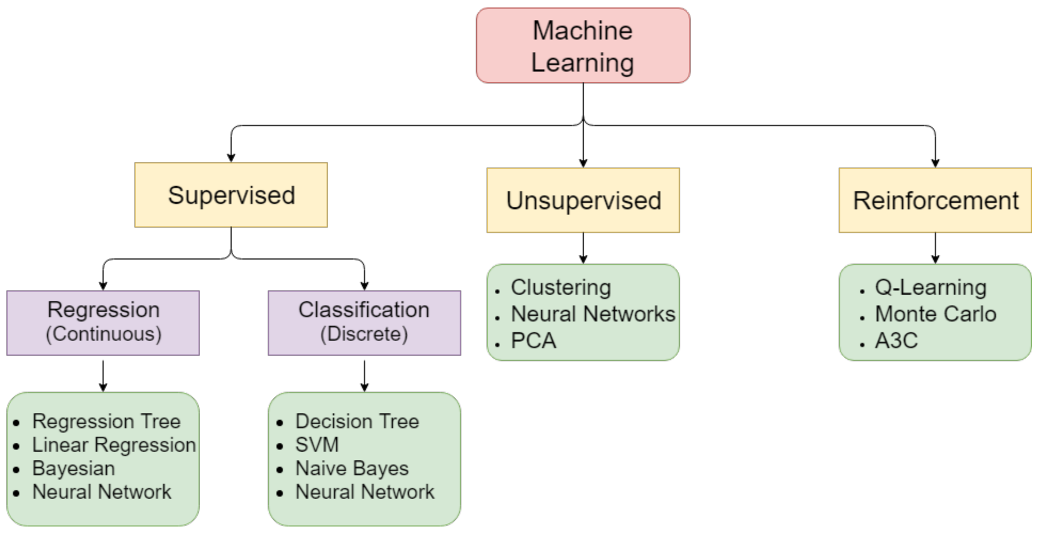


Figure 2.2: Branches of machine learning.

- Unsupervised machine learning is based around only having truthful input data. Computers do not know what the correct solution is to the data, but are trained to solve the underlying tasks based on different optimization functions. Unsupervised tasks can be tasks such as *clustering* where the computer is trained to find similarities in the given data and subsequently cluster into discrete groups.

- RL does not operate with the same type of input/output data scheme as the supervised/unsupervised learning, but is a way of learning where the computer will observe a *state* in a given *environment* and perform a certain *action*. The computer is trained by reinforcing correct behaviour, or action, and penalizing incorrect behaviour in any given state using a pre-determined reward function.

2.5.1 Neural Networks and Deep Learning

A common architecture used in ML is Neural Network (NN). An NN is a network of computational nodes consisting of three or more layers. The first layer in an NN is called the input layer, and the nodes in this layer contain the input data used during training. This data is in many cases pre-processed, e.g. via normalization, before being passed through the network. The last layer is called the output layer. In a network used for classification tasks, the output layer consists of as many nodes as there are discrete classes. In a regression task the output layer usually consists of only one single node, whose value will be continuous in the range of the output space. The layer, or layers, between the input and output layer is called the hidden layer, and consists of equations used to approximate a function, mapping the input values to the output value. To be able to train a neural network, each pair of nodes in the network is defined by a scalar *weight* used to influence how much the relationship between two nodes should impact the final output. During training these weights are gradually changed such that the full

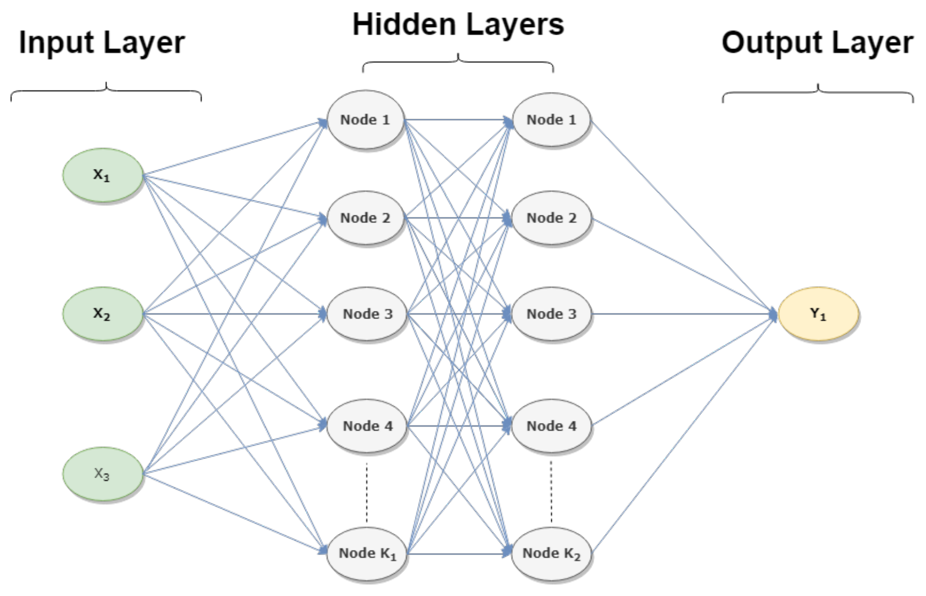


Figure 2.3: Feed-forward neural network.

network converges into a computational graph mapping the input values to the correct output.

Deep Learning (DL) is the process of stacking several layers of computation to increase the capabilities of the network. Most of the algorithms associated with the different branches of ML can be modeled as a deep architectures, and in many cases, such as NN and RL, this modeling is the norm. In a shallow architecture the input data would only go through a single computational process before being output. For deep architectures, the output of a computational process is passed to at least one more computational process before being output. A deep NN would for example have several hidden layers between the input and the output layer, such as in Figure 2.3. As is well known, deep learning has shown to produce better accuracy and performance over shallow learning on many tasks.

In regular NN architectures the data is processed in a *feed-forward* fashion, where the input data is computed in the first hidden layer, and the result in each node will be passed to corresponding nodes in the next layer. Figure 2.3 shows a feed-forward NN with 3 input nodes and 1 output node. The model has two hidden layers, and each layer *j* has K_j hidden nodes. The arrows in the figure represent how the computational output is passed throughout the network[30] [31]. Each blue line has an associated *weight* scalar, which are combined with a *bias* scalar in the node. During optimization, these scalars are changed according to an optimization function.

2.5.2 Recurrent Neural Network

Recurrent Neural Network (RNN)s are a type of NNs where the computation is not only performed in a feed-forward fashion, but additionally in a "time" dimension. The computed values in each time-step is passed as a *hidden state* to the next time-step. This property makes RNNs

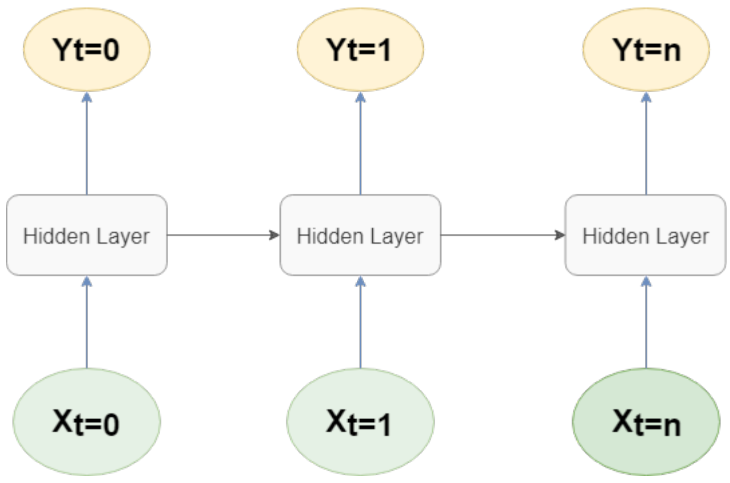


Figure 2.4: Recurrent neural network.

very suitable for modeling time-series and sequential data. The recurrent property ensures that the previous data is included as a "historical" feature when training and evaluating subsequent data. A regular feed-forward NN will have no concept of memory between two sets of input data, and is dependent on learning without any knowledge of what has occurred previously. In this thesis, the detected traffic is modeled as a time-series, with each access slot corresponding to a time-step.

Figure 2.4 shows a model of a single layer RNN. The input at each time-step is passed to the hidden layer, and the computational output of the hidden layer is then passed to the output layer. The computational output at time *t* is stored as a hidden state, which is passed to the hidden layer at time *t+1* and is used as an additional input feature [32].

2.5.3 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a special type of RNN used to improve learning over time. Regular RNNs are susceptible to problems such as vanishing or exploding gradients where the previous time-steps either affect the current computation too little or too much. LSTM nodes are often used to better handle the long-term dependency characteristic of time-series data. LSTM nodes include additional *gates* to control how much information should be included from the previous time-steps. During training the network will learn how much historical information is useful and ensure that the model is not overwhelmed with previous data. These gates have associated weights in the same manner as those between a pair of nodes. This causes LSTM nodes to be more complex than regular nodes and in turn makes the network able to learn more complex relationships [32].

2.6 Related Work

Congestion avoidance in LTE has been researched a lot, with both machine learning- and rule-based improvement schemes studied. 3GPP has released a list of improvement areas to help avoid congestion [15]. Improvements such as: Access Class Barring (ACB), Separate MTC RACH resources, Dynamic RACH resource allocation, MTC Specific Backoff, Slotted Access and Pull Based Schemes have been studied and considered as solutions to the congestion problem. ACB is used in LTE to give priority to certain classes, and assures that lower priority communication will wait longer in congested scenarios so that high priority traffic does not collide. Solutions such as ACB however only work as long as the eNB knows that there is congestion. The eNB will assume congestion if the QoS, or delay, crosses a certain threshold, but in those cases the congestion is already happening. Additionally, some improvements such as MTC Specific Backoff are most effective when the congestion level is low, and are less effective during large congestion [6]. An ideal scenario would be where the eNB can predict congestion in advance, and deploy the appropriate counter-measures, such as ACB or MTC Specific Backoff, before the traffic reaches its critical point.

Machine learning has previously been used to solve problems related to congestion issues in RACH with many promising solutions resulting from this research. However, most research has been focused on using machine learning to optimize the different "counter-measures" proposed by e.g. 3GPP [15] to help mitigate the congestion.

In [34] and [35] the authors use RL to dynamically choose the optimal parameters for ACB. ACB is useful to relieve congestion caused by bursty MTC traffic, and the RL based ACB approach achieves better results than using static ACB parameters. Both papers report success in dynamically changing parameters in response to bursty arrival traffic which reduces collision and improves throughput. ACB implements additional barring times and barring rates to devices that have yet to perform RA. The barring times are used to calculate additional backoff periods before transmitting the preamble. The barring rate is a number between 0 and 1 with increasing increments of 0.05, and is given to the devices according to their priority level. The device will, before RA, randomly draw a number in the range [0,1). If the number is higher than their barring rate, the device will wait for an additional backoff time calculated using the barring time.

A paper on detecting RACH collisions [36] looks at predicting the number of devices which selects the same preamble, by looking at the signal detected at the eNB. The paper is on the topic on preamble prediction and aims to predict preamble transmissions with two objectives: 1) check whether a certain preamble been sent; 2) check how many UEs have sent a specific preamble. The authors are able to predict the multiplicity of a certain preamble with consistently higher accuracy by using their NN approach over a logistic regression algorithm. The work does not look at predicting preambles at a future time-step, only at the current RA slot.

In [4], supervised machine learning is used to predict the number of *attempts* at each RA slot based on the detected preambles at the eNB. The eNB only has information about the number of successfully detected preambles at each RA slot, and has no information regarding how many devices have sent a preamble. During bursty conditions, the gap between detected and sent preambles increases substantially, and by predicting how many devices have sent preambles at each RA slot, the authors are able to determine whether the traffic is bursty or not. Using regression trees the authors are able to predict the number of attempts with a high probability. However, the authors only consider the case where the number of active devices are always 30000, and they don't test using an unknown number of active devices.

Chapter 3

Proposed Solution

This chapter first discusses assumptions and observations noted in this thesis. Then the proposed solutions are discussed at a general level, and the simulation data and parameters used are discussed. As stated in Chapter 1, two objectives are considered in this thesis. The first objective is to predict the traffic for future RA slots. Forecasting the traffic can help to predict traffic volume, which can be a benefit to both congestion classification as well as improvement selection. The second objective is to predict whether or not the traffic is leading to congestion.

3.1 Network Scenario and Assumptions

When doing simulations for this thesis, the following assumptions are made. The overarching work of predicting whether mMTC traffic will lead to congestion or not is a complicated task, so this thesis only considers certain scenarios.

The specific traffic condition in a real life scenario would be unbeknownst to the eNB. However, we only consider the bursty conditions in this thesis, and make no assumptions on the underlying traffic existing previous to the bursty traffic. Bursty traffic conditions occur when devices transmit data after being triggered by a certain event with a stochastic inter-arrival time. This unpredictability makes it difficult to plan for the bursty conditions in advance. As bursty traffic does not necessarily lead to congestion, this restricted scenario is still worth researching.

Further, the number of active devices, or state space, considered in this thesis is restricted to 10k, 20k, and 30k per cell. In this thesis, we also refer to the distinct number of active devices (10k, 20k, and 30k) as distinct *groups* of active devices. As the number of possible active devices can range from 0 to over 100 thousand, we propose a restricted scenario for our framework to simplify our task. Starting out in a restricted scenario allows us to more easily test our solution, and instead expand the scenario if our results are promising. These specific groups are selected as the traffic generated by them are different enough to produce distinct graphs in terms of attempts and detected attempts over time. For example, arrivals from more than 30k devices leads to congestion, while arrivals from less than 10k devices do not cause congestion. Arrivals from 20k devices are largely affected by the random factors in the access procedure, and do sometimes cause congestion.

The simulations shown in this thesis are slightly different from ones provided by 3GPP or other sources. In this thesis we do not consider channel impairments such as: path-loss, fading, inter-cell interference, etc. The use of different values for certain parameters will also create vastly different simulation results. For example, the PRACH config. index may considerably affect the number of attempts per slot, which in turn affects throughput.

Performance Measures	Arrivals	
	10000	30000
Collision Probability	1.98%	47.76%
Access Success Probability	100%	29.5 %

Table 3.1: Bursty traffic performance (LTE FDD) as mentioned in [15]

Table 3.1 shows performance during bursty conditions as measured by 3GPP. For 10k arrivals in a bursty window, the access success probability is 100%. In this case, the burstiness might affect the latency, but all arrivals will successfully connect to the eNB. With 30k arrivals, the access probability drops to below 30%, and nearly half of the attempts lead to collision. Based on these measurements, we assume that traffic from 10k devices do not lead to congestion, while for 30k active devices the traffic will lead to congestion. 3GPP did not perform measurements on the case of 20k active devices. We include 20k as a group of active devices in this thesis as it has some interesting results being midway between 10k and 30k.

As mentioned in Chapter 1, this work bases the arrival distribution on the models described by 3GPP in [15]. The technical report assumes that under bursty conditions, the access intensity at RA slot *i*, $A(i)$, is given as:

$$A(i) = N \int_{t_i}^{t_{i+1}} p(t)\, d(t)\,. \tag{3.1}$$

The integral is done over the interval $[t_i, t_{i+1}]$, where t_i is the time at access opportunity *i*, *t* is defined in the range $[0, T]$, and *T* is the observation window. $p(t)$ follows the beta distribution with parameters $\alpha = 3, \beta = 4$. N is the number of devices that activate during time T. For normal traffic, the arrival distribution follows uniform distribution over time T. For bursty and normal traffic, *T* is defined to be 10 and 60 seconds long, respectively.

RACH Parameter	Value
Number of contention-based preambles	54
PRACH config. index	6
Max. number of transmissions	10
Backoff value	20

Table 3.2: Simulation parameters for LTE RACH

Table 3.2 shows the parameters used in this thesis. The parameters not mentioned, e.g. response window size, are assumed in this thesis to not have any effect on the RA procedure. During RACH, a subset of the 64 total preambles are used for contention-free access and the rest are used for contention-based access. In this case, 10 preambles are reserved for contention-free RACH [15]. Using PRACH config. index 6, the eNB is open for RA every 5 ms. The maximum number of transmissions indicates the threshold before a device times out. After each failed transmission, the device will wait a random time between 0 and the backoff value counted in milliseconds. In this study the maximum backoff would be 4 RA slots. These parameters are similar to the ones used by 3GPP in [15]. Because we disregard any channel impairments as well as any contention resolution, we assume that the probability of a device successfully completing the RA procedure is 1 when the device uniquely selects a preamble, and 0 otherwise.

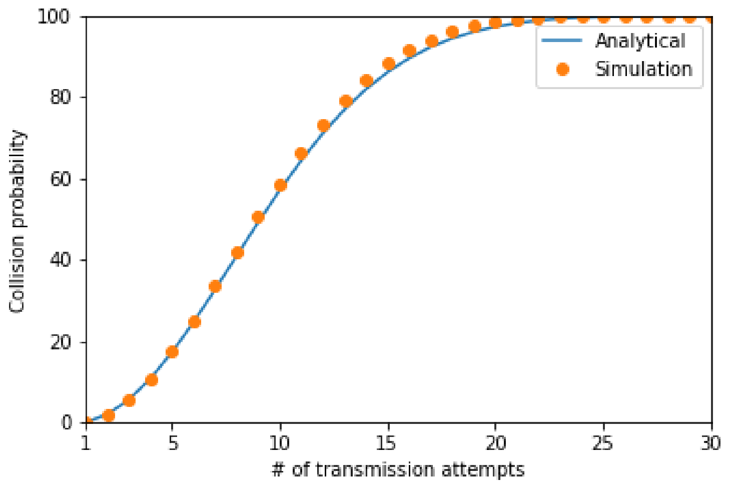


Figure 3.1: Probability of at least 1 collision in a given RA slot using 54 preambles.

Figure 3.1 shows the probability of at least one collision occurring during an access opportunity for 54 available preambles. The probabilities are calculated based on a generalized version of the “Birthday Problem” [38]. We define the stochastic variable C as the number of collisions in one RA slot. $P(C \geq 1)$ is the probability of at least 1 collision, which is equal to

$1 - P(C = 0)$. We calculate $P(C \geq 1)$ for a given *n* using the formula in Eq. 3.2:

$$P\left(C \geq\ 1\right)\ =\ 1\ -\ \left(\frac{d-1}{d}\right)^{\frac{n(n-1)}{2}}. \qquad (3.2)$$

where d is equal to 54. We define *n* as the number of attempts in on RA slot, and d as the number of available preambles. The calculation is verified via simulation. We observe in Figure 3.1 that the probability of collision exceeds 50% at around 9 transmission attempts per slot. For 20 attempts per slot the collision probability is 97%.

3.2 Proposed Solutions

The simulation result illustrated in Figure 3.1 highlights the rapid increase in collision probability following only a few additional transmission attempts per slot. Predicting transmissions attempts in the future will assist the eNB in estimating the significance of future collisions, and help predict whether or not the network will be congested.

The two solutions proposed in this thesis function in addition to the congestion avoidance schemes previously studied by other sources. Figure 3.2 illustrate the additional steps of predicting and classifying the traffic in the decision process. The additional steps this work is contributing with are marked in red, with the numbers corresponding to the objectives in Chapter 1. The first solution considers the objective of forecasting traffic arrival. The second solution considers the objective of classifying mMTC traffic as leading to congestion.

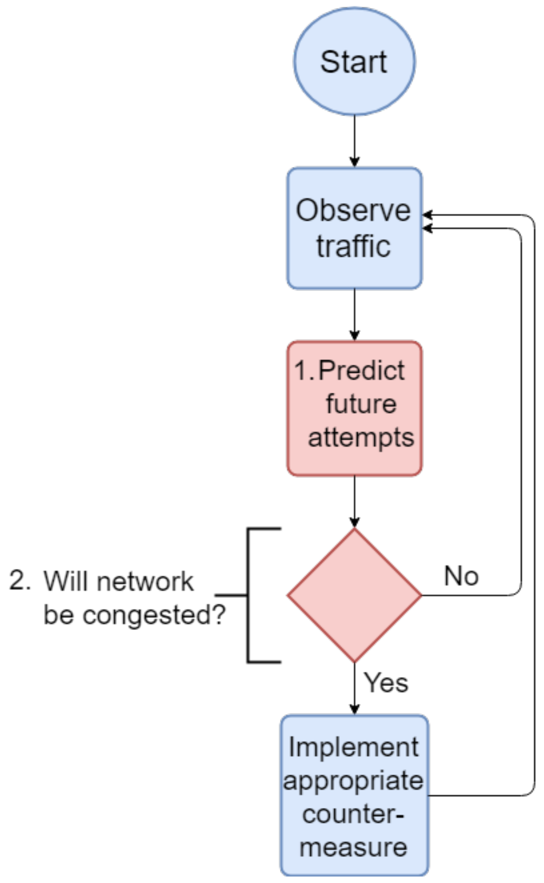

Start
Observe traffic
1. Predict future attempts
2. Will network be congested?
No
Yes
Implement appropriate counter-measure

Figure 3.2: Overview of solution framework.

3.2.1 Future Prediction Model

For the task of predicting future detected attempts, the solution is structured as a multi-step prediction task. A common approach to multi-step prediction is called *recursive prediction*. In this approach an *input seed* of real values are used as basis for new, predicted values. The input seed is a set Y of n input values, $Y = (y_1, y_2, ..., y_n)$, used to predict the next value, denoted as $\hat{y}_{n+1}$. In order to predict the value after, e.g. $f(Y) = \hat{y}_{n+2}$, the set of input values will include the previous predicted output so that $Y = (y_1, y_2, ..., y_n, \hat{y}_{n+1})$. This approach can be done as a sliding window, where $|Y|$ is constant, or an increasing window where $|Y|$ increases for each prediction.

$$\begin{aligned} t_0 &: f(y_1, y_2, ..., y_n) = \hat{y}_{n+1}, \\ t_1 &: f(y_2, y_3, ..., \hat{y}_{n+1}) = \hat{y}_{n+2}. \end{aligned} \tag{3.3a}$$

$$\begin{aligned} t_0 &: f(y_1, y_2, ..., y_n) = \hat{y}_{n+1}, \\ t_1 &: f(y_1, y_2, ..., y_n, \hat{y}_{n+1}) = \hat{y}_{n+2}. \end{aligned} \tag{3.3b}$$

Eq. (3.3a) shows the function for a fixed sliding window, while Eq. (3.3b) shows the function with an increasing window. Note that for t_1 in Eq. (3.3a) the set *Y* no longer includes y_1.

The drawback of using a fixed sliding window is that after *n* steps the new predictions are purely based on previous predicted values, while an increasing window will keep the initial real values at all times. However, increasing the window each step will cause the subsequent predictions to have increased computational complexity.

The proposed model uses an LSTM network to perform multi-step prediction. With this architecture, the model has the advantages of both fixed and increasing window functions. The *hidden states* in an LSTM keeps track of the previous computations at the current time-step, such that when predicting the value at time $t+1$, the function uses only the summarized values stored in the hidden states. This ensures that the model always uses the information from the original real values when predicting, but does not suffer from increased computational complexity.

The model accepts a series of inputs, $X = (x_0, x_1, ..., x_n)$, and returns an output, $Y = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_{n+1})$. Figure 3.3 shows how the evaluation for the next time-step is based on the previous time-steps. At each time-step, the model takes in a single input feature, the value at time *n*, which is combined with the *hidden state* from the previous time-steps, to compute a single output, the value at time *n+1*. E.g. for predicting value $\hat{x}_{n+3}$, the first *n* values are the real detected values, while $\hat{x}_{n+1}$ and $\hat{x}_{n+2}$ are previous predicted outputs.

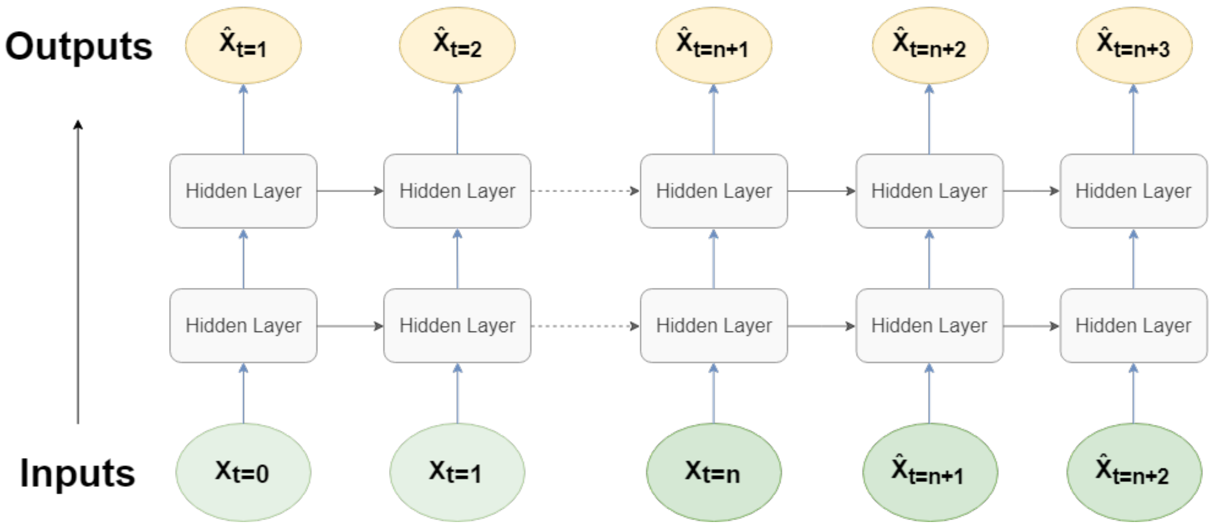


Figure 3.3: A two layer LSTM used for multi-step prediction by predicting one step at a time.

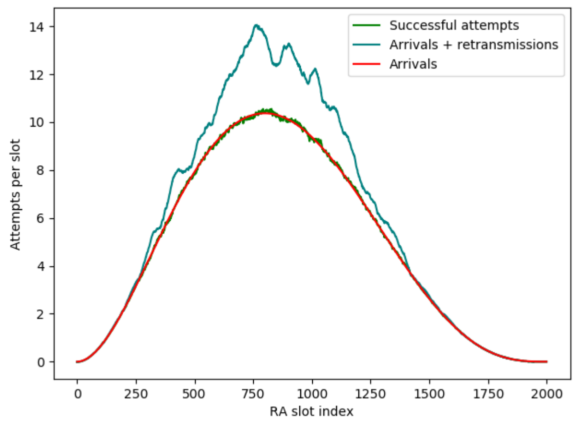
3.2.2 Congestion Classification

Although the mMTC traffic may indicate a bursty scenario, the volume of arrivals might not cause a congestion in the network. We propose therefore a NN based classification model to predict if congestion is likely to occur in the bursty window. As the data available is relatively low complexity in only two dimensions, using a low complexity classification algorithm such as a feed-forward NN should be sufficient for classifying the data.

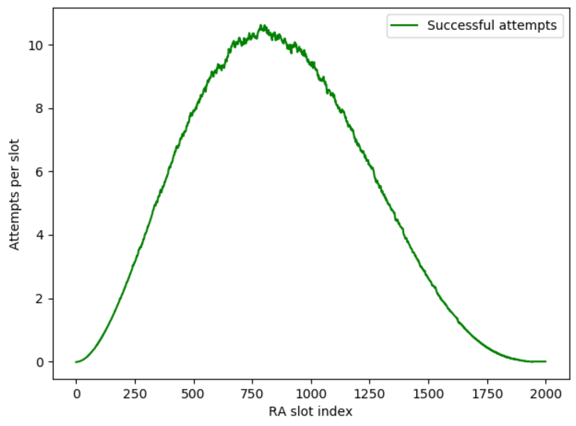
Figure 3.2 shows the pipeline starting from the detected data available at the eNB to the classified congestion output. The output from the prediction model, which includes both the original input as well as the predicted future, is used as input for the classification model.

3.3 Traffic Pattern Analysis

This section studies the relevant traffic conditions considered in this thesis. The traffic in the following figures are simulated based on the assumptions stated in Section 3.1, and illustrate the relationship between mMTC devices attempting to connect with the eNB, and the number of successful attempts.

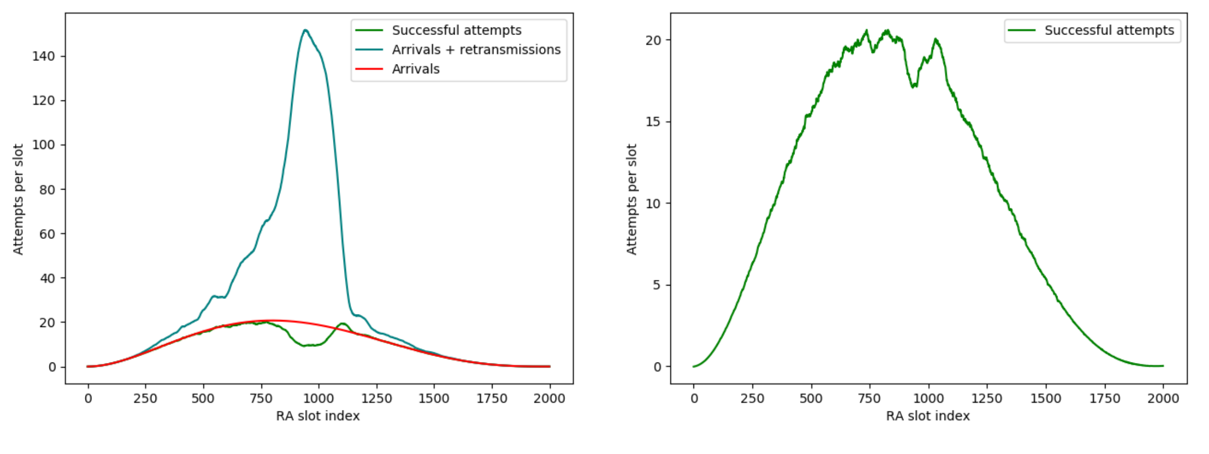


(a) Arrivals and attempts.



(b) Successful attempts.

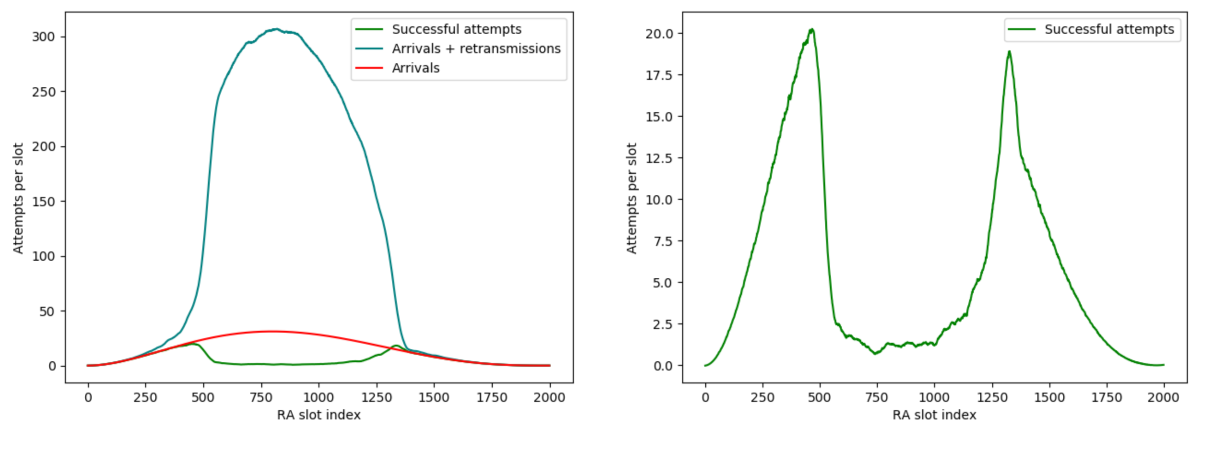
Figure 3.4: Bursty traffic patterns for 10000 active devices.



(a) Arrivals and attempts.

(b) Successful attempts.

Figure 3.5: Bursty traffic patterns for 20000 active devices.



(a) Arrivals and attempts.

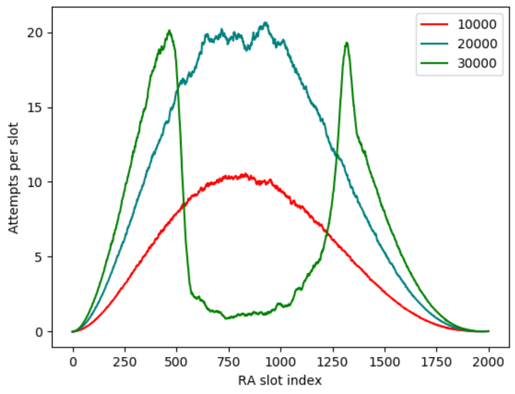
(b) Successful attempts.

Figure 3.6: Bursty traffic patterns for 30000 active devices.

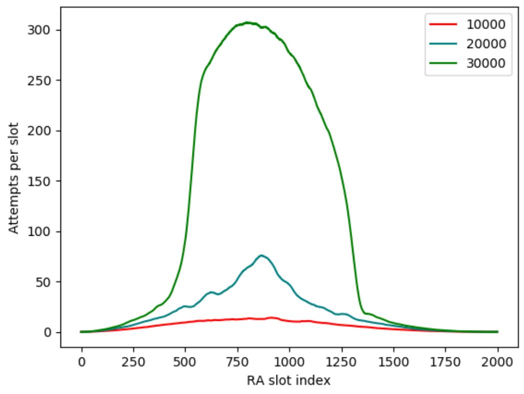
The traffic in Figures 3.4, 3.5 and 3.6 are simulated based on the parameters listed in Table 3.2. The data is then smoothed through a Savitzky-Golay [37] filter with a window size of 97 and a 2nd degree polynomial. This filtering mechanism is done to create continuous values from the discrete simulation output. Smoothing values is a common step to improve results when using machine learning on time-series data.

Figure 3.4(a) shows the traffic pattern for 10k devices during bursty conditions. Although there is some retransmission, the detected attempts tightly follow the arrival distribution, and all attempts are successful. Figure 3.4(b) highlights the detected attempts which is used as input data during training. We observe in Figures 3.4(a) and 3.5(a) that the eNB does not experience meaningful congestion even though some collision occur. In both scenarios the detected attempts are able to tightly follow the arrival distribution. Note that the y-axes have different limits in the two scenarios.

Figure 3.6 shows arrivals based on 30k active devices. We observe that congestion starts occurring around RA slot index 450, after a massive spike in retransmission and arrivals. Figure 3.6(b) highlights the drastic reduction in detected attempts as soon as congestion occurs. The cascading collisions and retransmissions keep the congestion in place until enough individual devices time out.



(a) Detected attempts for 10k, 20k, and 30k active devices.



(b) Arrivals + retransmission for 10k, 20k, and 30k active devices.

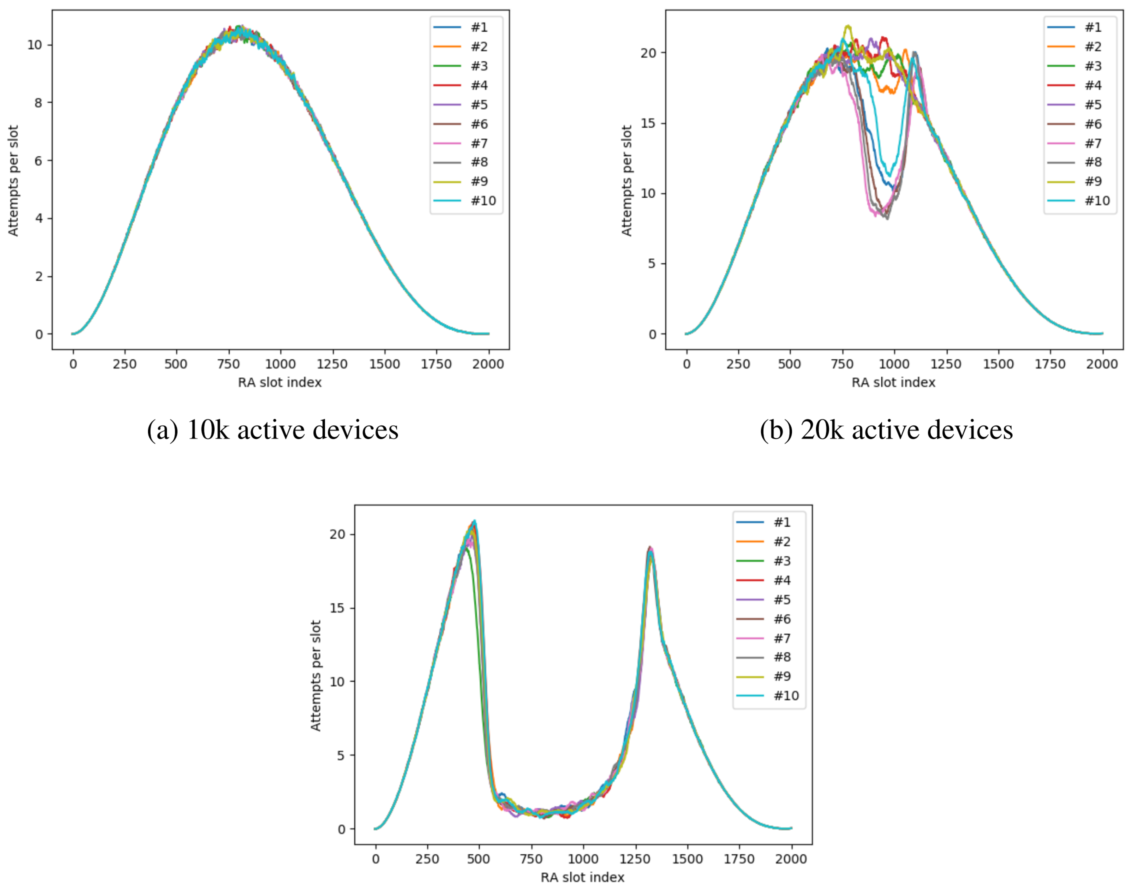
Figure 3.7: Comparison of RA attempts.

Active devices	Detected / Arrivals in Fig. 3.7(a)	Avg. ratio
10000	100%	100%
20000	99.26%	95.72%
30000	37.64%	37.23%

Table 3.3: Ratio of detected attempts vs arrivals during RA

The results in Table 3.3 shows the ratio of successful vs actual RA procedures, corresponding to Figure 3.7. The average ratio is calculated by measuring the number of successful attempts (detected by the eNB) divided by the number of new arrivals, for 100 simulations. By comparing the successful arrivals based on Figure 3.7(a) with the average ratio, we observe that there is little difference for 10k and 30k, while for 20k the number of successful arrivals differ some from the average.

In Figure 3.7 we see how the different traffic patterns compare. For the case of both 20k and 30k active devices, the maximum number of attempts in an RA slot is approximately the same, at around 20 attempts per slot. However, for the case of 30k devices, the larger volume causes congestion in the network. In Figure 3.7(b) we observe the large difference in attempts for each 10k increment in active devices.



(a) 10k active devices

(b) 20k active devices

(c) 30k active devices

Figure 3.8: Successful attempts per slot for 10 different simulations of 10k, 20k, and 30k active devices.

Figure 3.8 shows the different simulation results corresponding to detected arrivals for the three groups over 10 trials. For 10k and 30k, the simulations produce more or less identical plots with little difference. The 20k group shows considerable difference with almost half the simulations experiencing congestion, and half experiencing very little congestion. This can be attributed to the randomness during preamble selection. We observe that the average access success probability for 20k devices, as noted in Table 3.3, is over 95%, even though some congestion occur during simulation. Because the average access success probability is so high, we assume that any congestion caused by 20k devices is insignificant, and thus for the following tests in Chapter 4 we assume that traffic generated by 20k devices does not lead to congestion.

Chapter 4

Algorithm Implementation and Model Evaluation

In this chapter, the framework created based on the proposed solutions is presented. Then, the scenarios and cases considered during testing are listed. This chapter is divided into three scenarios, focusing on different traffic conditions and objectives. Scenario 1 focuses on objective 1, while Scenarios 2 and 3 studies objective 2. Each of the scenarios are further divided into smaller cases. The results from the test scenarios are then detailed and discussed, and the previous chapters are summarized with regards to the results.

4.1 Algorithm Overview

The proposed solution framework presented in Chapter 3 consists of three separate algorithms. The first algorithm details the *data simulation phase*, simulating traffic data in the network and at the eNB. The second algorithm uses the simulated traffic from the first algorithm as input to predict the future traffic arrival. The third algorithm uses both the simulated traffic from algorithm 1, and the output of the second algorithm, to classify the traffic and identify congestion status. All code used in this thesis is written in Python (v3.6.7) using the PyTorch (v1.0) library for machine learning.

Algorithm 4.1 Data generation

```
Input: Cell population
Output: Smoothed data on network traffic
for 1:Number of Active Devices do
    Generate arrivals per slot;
    Simulate collisions;
    Smooth with Savitzky-Golay filter;
    Return smoothed data;
end for
```

Algorithm 4.2 Traffic prediction

```
Input: Output from algorithm 4.1
Input: (n, k), n>>k
Input: Time-series length $L$
Output: Predicted traffic
Choose $k$ samples from Input as test data;
Choose $n - k$ samples as training data;
Training input = training data [0:$L$-1];
Training output = training data [1:$L$];
Train model with training data;
loop K Samples in Test Set
    for 1:Number of Future Slots do
        Feed test data into trained model;
        Predicted next step value is extracted from tested model;
        New test data is previous test data + predicted next step value;
    end for
    Return predicted traffic;
end loop
```

Algorithm 4.3 Congestion prediction

```
Input: Output from Algorithm 4.1
Input: Output from Algorithm 4.2
Output: Binary classification
Label data from Algorithm 4.1 as congestion/ not congestion;
Train model on labeled data;
loop K samples from Algorithm 4.2
    Test model on sample;
    Return prediction;
end loop
```

4.2 Test Scenario Overview

Scenario 1 considers the multi-step prediction model for traffic forecasting. This scenario is divided into cases affiliated with either a *homogeneous* case, Case 1A, or a *heterogeneous case*, Case 1B. In Case 1A, the multi-step prediction model is trained on a single group independently. This trained model is tested on data from the respective group, and the model thus predicts the future attempts in a *static* scenario where the number of active devices in a cell does not change. In Case 1B, the heterogeneous case, the multi-step prediction model is trained on multiple groups simultaneously, e.g. 10k & 20k, 10k & 30k, etc. The model is evaluated by predicting the attempts in a *dynamic* scenario where the number of active devices is unknown, but belonging to one of the relevant groups. This implies the model has to learn to separate the groups within the network. This is a more challenging problem than Case 1A, but subsequently more akin to a real-life scenario, where the number of active devices in a cell is dynamic. For both cases we forecast the arrivals up to the ith access slot, with i equal to 1000 and 2000. These values are selected as the bursty window has in total 2000 access slots, and we observe that after approx. 1000 slots the predictions become less accurate.

Scenarios 2 and 3 consider the objective regarding congestion classification. In Scenario 2, the forecasted traffic from Case 1B is studied, and we look at two cases with regards to the forecasted traffic length. Case 2A considers traffic forecasted to the 1000th slot, while Case 2B considers traffic forecasted to the 2000th slot.

Scenario 3 studies only traffic data available at the eNB, i.e. non-forecasted traffic, for predicting congestion. This is in contrast to Scenario 2, where forecasted traffic data is used. Comparing the results in Scenarios 2 and 3, we can conclude whether or not the additional step of forecasting traffic is beneficial for predicting congestion. We again divide the scenario into two cases. The first case, Case 3A, uses the input seeds from Case 1B, which are based on traffic from either 10k, 20k, or 30k active devices. The second case, Case 3B, uses traffic from randomly selected number of active devices between 10k and 30k. Because the classification task is a simpler task than multi-step prediction, we are able to consider a more general state space than in Scenarios 1. We evaluate the trained models in both Scenarios 2 and 3 by predicting if the network is approaching congestion or not. As we assume in Chapter 3.3 that certain groups either always lead to congestion or do not lead to congestion, both scenarios consider a binary classification task.

Table 4.1 shows the scenarios and cases considered. For all scenarios we consider different length input seeds during testing: 100, 200, 300, and 400. The maximum input seed length is set as 400, as bursty traffic starts to congest around this access slot for traffic based on 30k active devices and more.

Scenario	Case	Explanation	State space
1	Case 1A: Homogeneous forecasting.	Trained and tested on a single group.	{1}
	Case 1B: Heterogeneous forecasting.	Trained and tested on multiple groups.	{2}
2	Case 2A: Classify traffic to the 1000th slot.	Uses predicted traffic from Case 1B.	{2}
	Case 2B: Classify traffic to the 2000th slot.		
3	Case 3A: Classify original input seeds.	Uses the same input seeds as in Case 1B.	{2,3}
	Case 3B: Classify new input seeds	Random traffic between 10k and 30k.	{20000}

Table 4.1: Overview of scenarios and cases

4.3 Results for Test Scenario 1

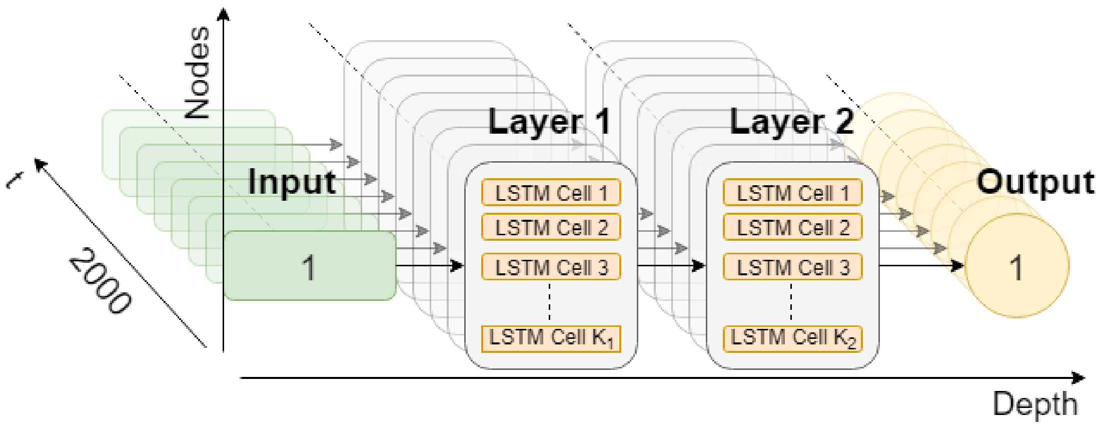


Figure 4.1: Model of the traffic prediction network.

The prediction network, show in Figure 4.1, consists of an input layer with a single input node, two LSTM layers, and an output layer with a single output node. The model has one input feature, the value at the current time-step, and one output value, which is the value at the next time-step. During training, the full bursty window is used, and the model has therefore a size of 2000 in the "time" dimension. During testing, we use a shorter input seed. Drawing an analogy to a three-dimensional Cartesian system, the number of layers in the model is represented along the X-axis, the number of nodes per layer along the Y-axis, and the time-steps used when evaluating is represented along the Z-axis. Each node in the *hidden layers*, shown in Figure 4.1 as LSTM cells, have a corresponding *hidden state* which is passed along the network in the Z-dimension, i.e. through time. Each hidden layer j has K_j number of LSTM cells, or hidden nodes. Each case in the scenario is trained on a separate model, and the number of hidden nodes in each model is not necessarily the same. Different cases have different complexity, and might require a different sized model.

For both cases in Scenario 1, the MSE is calculated between the predicted and forecasted traffic. The MSE is averaged over 20 trials. We use 20 trials as the MSE starts converging at this point.

Model parameter	Value
Hidden layers	2
Loss function	MSE
Optimization function	Adam [39]
Learning rate	0.0001
Activation function	Tanh [40]
Dropout value	10%

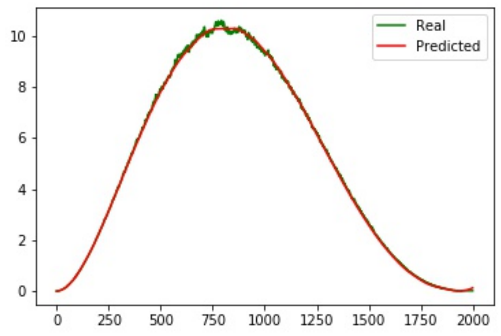
Table 4.2: Common model parameters and functions used in all experiments

4.3.1 Test Results for Case 1A

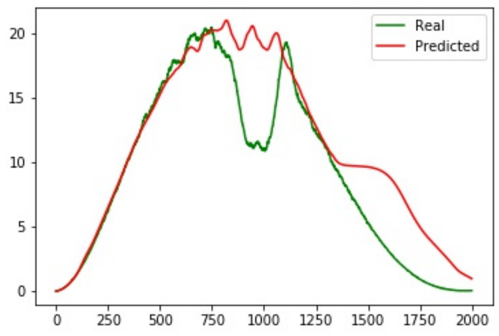
To assure that the model is able to learn the features of the data set, it is first trained to overfit on each group separately, referred to as the homogeneous case. As mentioned in previous chapters, we consider three groups of active devices: 10k, 20k, and 30k. We train the model on each of these three groups separately, using the parameters which works best on each individual group. Based on the results from this case, we get an intuition on how to configure parameters to use for the heterogeneous case, and we can use the trained models from Case 1A to improve training in Case 1B via transfer learning.

Group	Seed Length	MSE	
		$i = 1000$	$i = 2000$
10k	100	0.0070	0.0053
	200	0.0065	0.0055
	300	0.0059	0.0066
	400	0.0054	0.0085
20k	100	5.7293	8.4738
	200	5.8040	8.7326
	300	5.7614	8.5462
	400	5.6758	24.7791
30k	100	1.9281	1.1734
	200	2.5648	1.3886
	300	1.7874	1.1598
	400	0.4937	1.5370

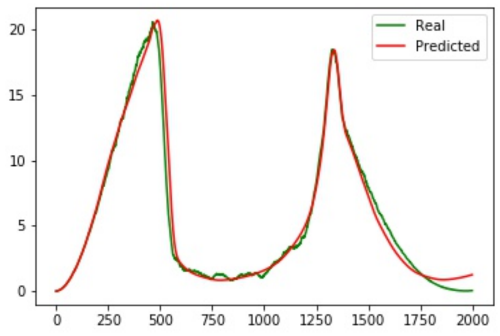
Table 4.3: Effect of using different input lengths measured in MSE between real and predicted traffic



(a) 10k with 1000 hidden nodes



(b) 20k with 1000 hidden nodes



(c) 30k with 1100 hidden nodes

Figure 4.2: Multi-step prediction using 100 RA slots as input seed.

Because of the low collision probability for 10k devices, the simulated sequences of detected attempts are nearly identical. This is illustrated in Figure 4.2(a) and 4.2(c), where we observe that the model easily learns to fit the data. Figure 3.8 in Chapter 3 also reflects this behaviour. The large collision probability for 30k devices causes those simulations to act in a similar manner as well.

Figure 4.2(b) shows the model is worse at fitting the data for 20k. As shown in Figure 3.8(b), the traffic for 20k which leads to congestion has almost the same curve in the beginning as the traffic not leading to congestion. When using at most 400 slots as input, it is difficult for the model to distinguish this behaviour, as it is so unpredictable. Another item to note is the good fit in the beginning, before the models decrease in accuracy. This can imply that the models are not effective for long-term predictions with complex data.

Table 4.3 shows how well the model is able to fit to the various homogeneous groups measured in MSE. Based on the results shown in Table 4.3, we observe that the seed length does not affect the MSE or the performance of the model much for this test case. The predicted distance does have some impact, such as for 20k active devices. This is illustrated in Figure 4.2(b) where the predictions deviate after slot 1250.

The results in Case 1A suggest that the model is able to overfit on the easier patterns of the homogeneous groups such as 10k and 30k, while 20k is too complex to easily overfit. In Figure 3.8 from Chapter 3, we observe that the detected attempts in the first 400 slots are almost indistinguishable for each simulation, and thus it is difficult for the model to learn if the traffic is leading to congestion or not. As mentioned in Section 4.2, we do not consider input seed lengths of more than 400, as the congestion starts occurring at approximately slot 450 for 30k devices. Using a longer seed would no longer give the eNB any advantage, as the congestion will in certain conditions

The results on the homogeneous case lead us to conclude that the time series are not too complex for the network to learn. We thus claim it is possible for the network to learn the heterogeneous case as well. already have occurred. Based on the results from the homogeneous case, it is reasonable to assume that the heterogeneous case will not be able to confidently predict further than approx. 1000 RA slots.

4.3.2 Test Results for Case 1B

The task of prediction in the heterogeneous case is more challenging than in the homogeneous case. In this case, we study the effects of different grouping scenarios on the performance. We refer to a group of several active devices as a Heterogeneous Group (HG). The model is trained independently on each HG, and tested with input seeds from each of the respective sub-groups. The predicted output is then compared with true arrivals from the correct sub-group. Based on the results achieved in Case 1A, we utilize a transfer learning technique to speed up training for Case 1B. Transfer learning reuses an already trained network on a new task to give the network pre-trained weights when training on the new task. For this case, the model already knows the general pattern for 10k, 20k, and 30k when using their respective trained models for transfer learning. The trained models chosen are determined based on which one performs best during training on the heterogeneous group.

Three HGs are considered:

- HG1: 10k & 20k.
- HG2: 10k & 30k.
- HG3: 20k & 30k.

Tables 4.4-4.6 show MSE per HG calculated to the *i*'th slot with four different input seed lengths.

Sub-Group	Seed Length	MSE	
		$i = 1000$	$i = 2000$
10k	100	4.1854	3.0095
	200	1.5812	1.4973
	300	0.8210	1.0292
	400	0.2121	0.5782
20k	100	3.8421	61.8212
	200	5.7253	62.7364
	300	4.9141	62.5750
	400	3.2845	61.5249

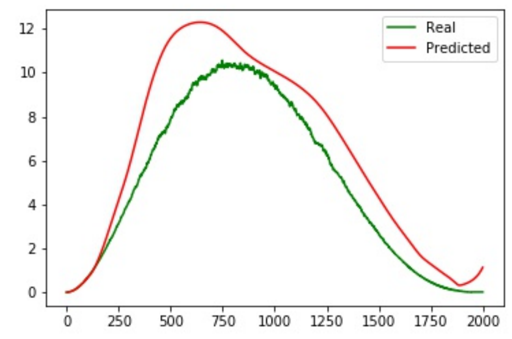
Table 4.4: HG 1 with MSE

Sub-Group	Seed Length	MSE	
		$i = 1000$	$i = 2000$
10k	100	2.0010	38.6759
	200	5.3938	23.6568
	300	10.8800	25.3231
	400	13.9169	23.3842
30k	100	33.7118	22.6454
	200	1.6859	6.8460
	300	2.2653	7.1435
	400	0.8487	5.2137

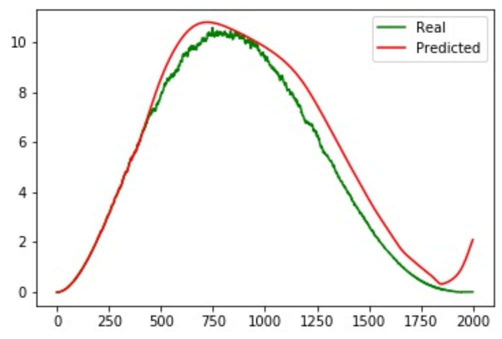
Table 4.5: HG 2 with MSE

Sub-Group	Seed Length	MSE	
		$i = 1000$	$i = 2000$
20k	100	111.8148	109.6883
	200	101.8806	109.7299
	300	45.3941	54.9247
	400	25.1175	44.8306
30k	100	142.7067	102.3264
	200	112.3287	83.5336
	300	12.2027	37.2768
	400	11.2551	21.6182

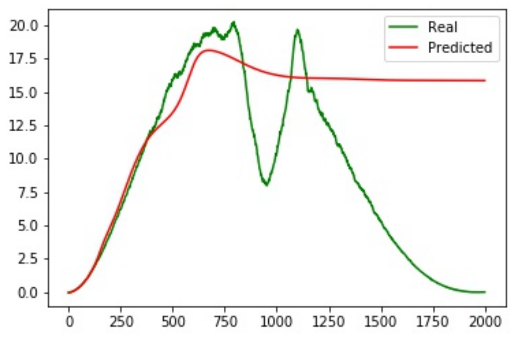
Table 4.6: HG 3 with MSE



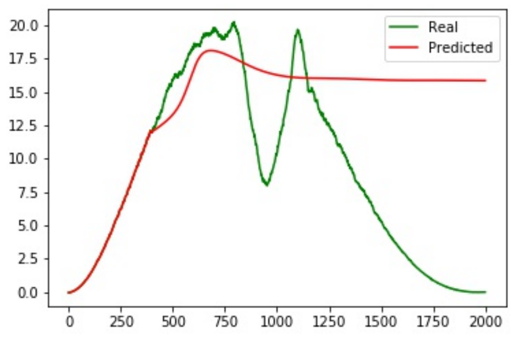
(a) MSE=3.0 for 10k with 100 slot input



(b) MSE=0.5 for 10k with 400 slot input

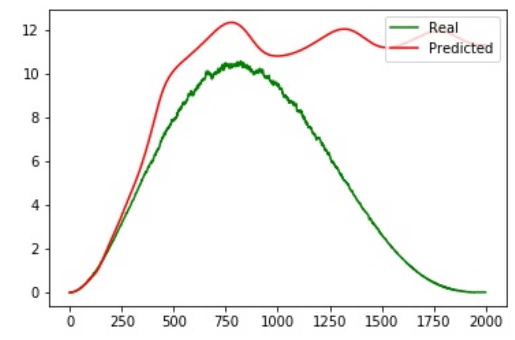


(c) MSE=61.8 for 20k with 100 slot input

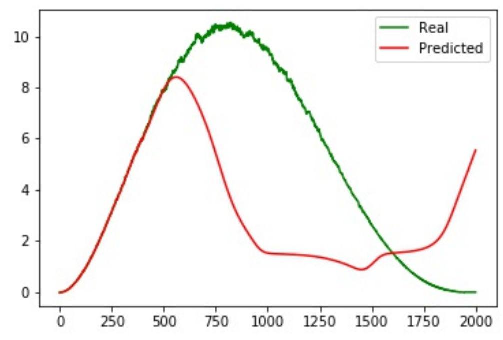


(d) MSE=61.5 for 20k with 400 slot input

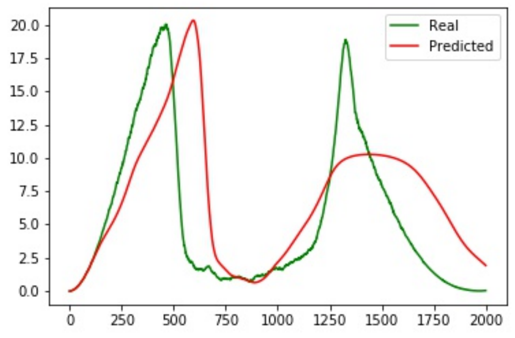
Figure 4.3: Predictions from HG 1 visualized with different length input seeds.



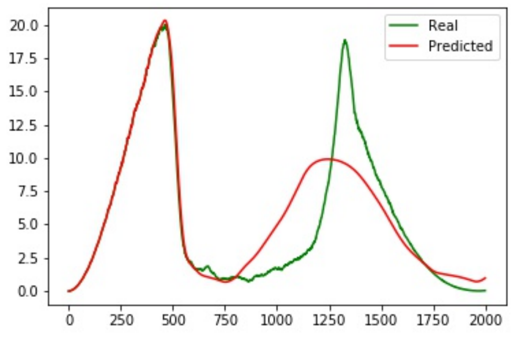
(a) MSE=38.6 for 10k with 100 slot input



(b) MSE=23.4 for 10k with 400 slot input

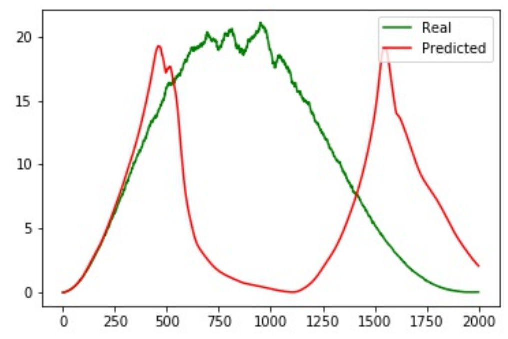


(c) MSE=22.6 for 30k with 100 slot input

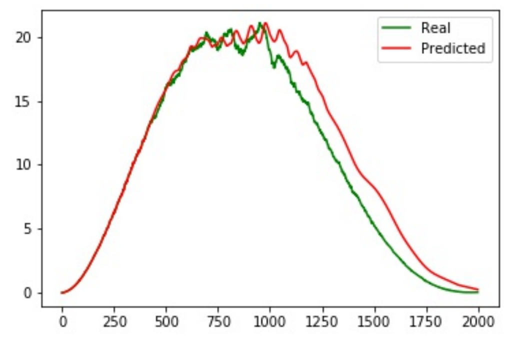


(d) MSE=5.2 for 30k with 400 slot input

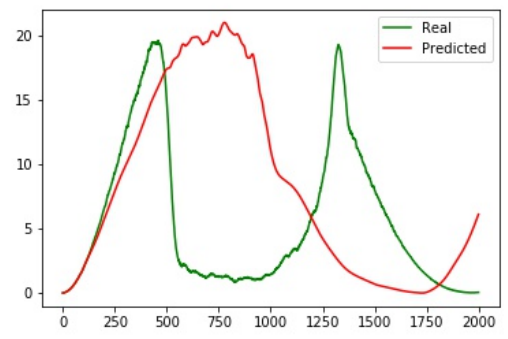
Figure 4.4: Predictions from HG 2 visualized with different length input seeds.



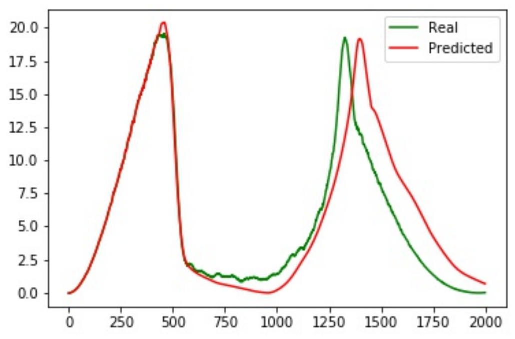
(a) MSE=109.7 for 20k with 100 slot input



(b) MSE=44.8 for 20k with 400 slot input



(c) MSE=102.3 for 30k with 100 slot input



(d) MSE=21.61 for 30k with 400 slot input

Figure 4.5: Predictions from HG 3 visualized with different length input seeds.

We visualize in figures 4.3(a) and (b) how the MSE for HG 1 is affected by using a longer input seed. With a longer seed, the model has more real information from the beginning of the prediction, and is able to follow the true traffic arrival more closely. This is reflected in the first row of Table 4.4 with *i* equal to 2000, and with seed length 100 and 400, respectively. Figures 4.3(c) and (d) illustrate a different behaviour where the model is not able to better predict the arrivals with a longer input seed. However, it is able to follow the traffic arrival almost to the maximum value before it flattens out.

For HG2 we have similar behaviour, but in opposite fashion for the 10k group. The model does not predict more accurately with a longer input seed, whereas in HG1 the 10k group improves with the longer input. The 30k group in HG2 does however get a reduced MSE when using a longer input seed.

For HG3, the model performs worse than on the other two HGs. A factor for this could be that HG3 consists of the 20k and 30k sub-groups, which have very similar arrival patterns before the congestion occurs. Both sub-groups peak at around 20 arrivals between the 500th and 750th slot. As the model is more precise in predicting arrivals in the beginning, the similarity between these sub-groups so early might cause the model to under perform compared to the other HGs. Figure 4.5 illustrates the mediocre performance when only using 100 slots as input seed. If we increase the input seed, the model is able to differentiate the two sub-groups, and the MSE decreases. For HG3, an input seed of length 300 is needed to be able to forecast the traffic arrival with satisfactory error, as can be seen in Table 4.6 column 3, where the MSE is reduced from 109.7 and 102.3, to 54.9 and 37.2, for the respective sub-groups.

We observe in the tests from Scenario 1, that although the traffic prediction model is not able to produce consistent and accurate long-term traffic forecast, it does manage to forecast the general trend of the traffic, and is in many cases able to follow the shape of the real traffic arrival. We note that using a input length over 100 does in some cases have a meaningful effect on the model performance and results for traffic prediction. For the HGs with similar sub-groups, the MSE decreases when using a longer input seed. This is expected as the model has more truthful information to base the predictions on. We also observe that the models tend to be accurate for the first half of the bursty window, and thereafter lose most of its veracity.

We conclude that for the HGs where the sub-groups are easier to distinguish, the model is able to forecast the traffic arrivals based on only 100 input slots, and using a longer input seed in these cases does not seem to improve the prediction much. For HG3 where the sub-groups are more difficult to distinguish, the model requires a longer input seed to be able to forecast the traffic adequately.

4.4 Results for Test Scenario 2

In this section, both the model architecture and the results for the second scenario are presented. The classification model is a simple single-layer feed-forward NN with a logistic function, also referred to as the sigmoid function, in the output layer.

$$f(x) = \left(\frac{1}{1+e^{-x}}\right). \tag{4.1}$$

Eq. 4.1 shows the sigmoid function. The output of the sigmoid function is a value in the range [0,1], and this function is often used to calculate probabilities. The output of the model is the probability of the traffic belonging to a certain class. The classes will in this case be *congestion* and *no congestion*.

Figure 4.6 illustrates the feed-forward NN. The network has *K* input nodes, corresponding to the number of input values, e.g. 100, 200, 300, or 400. The network has two output nodes, each corresponding to one class.

The model is trained on the simulated traffic arrivals illustrated in Chapter 3, and tested on the forecasted traffic from Case 1B. We consider two test cases with respect to the length of the forecasted traffic. Case 2A uses *forecasted* traffic to the 1000th slot, while Case 2B

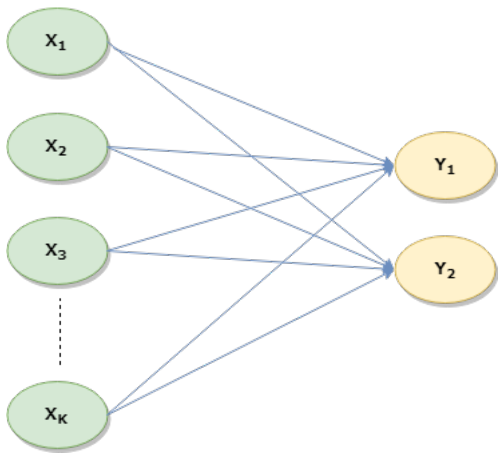


Figure 4.6: Classification model.

uses forecasted traffic to the 2000th slot. We separate these two cases to study whether or not the length of the forecasted traffic has any significant effect on the classification task. As this scenario uses the traffic generated from Case 1B, we consider the same HGs as in Scenario 1. Although we define our task as a binary classification task for predicting congestion or not, HG 1 consists of two sub-groups where neither leads to congestion. However, we still choose to consider this HG in our test scenarios, as it gives more insight into the differential abilities of the proposed solution framework.

Tables 4.7-4.9 show the test accuracy for the different HGs with various length input seed.

Seed Length	Accuracy	
	Test Case 2A	Test Case 2B
100	50%	50%
200	50%	50%
300	50%	50%
400	100%	100%

Table 4.7: HG 1 accuracy

Seed Length	Accuracy	
	Test Case 2A	Test Case 2B
100	100%	100%
200	100%	100%
300	100%	77.5%
400	100%	100%

Table 4.8: HG 2 accuracy

Seed Length	Accuracy	
	Test Case 2A	Test Case 2B
100	0%	5%
200	10%	10%
300	80%	80%
400	90%	90%

Table 4.9: HG 3 accuracy

The results in Scenario 2 illustrate that predicting congestion based on forecasted traffic performs better than random guessing in most of the tests. For HG 1, the model achieves 100% when using an input seed of length 400, but for shorter input seeds the model performs on par with random guessing. We observe in Table 4.7 that the input seed length is more significant in predicting congestion than the length of the forecasted traffic, which seemingly has no significant impact. On HG 2, the model is able to achieve 100% accuracy in most tests. In the case of HG 3, the forecasted traffic, as illustrated in Figure 4.5, initially follows the wrong arrival pattern, which is reflected in the low accuracy when using input seed of lengths 100 and 200. However, when using a longer input seed the model is able to predict the correct class with a very high accuracy.

The results show that the model is able to correctly predict congestion based on forecasted traffic assuming the input seed is long enough. However, using 400 slots as the input seed is reaching the limit for how close to the congestion the model can accurately predict. In the cases where congestion starts around slot 450, the proposed framework will only have 50 slots to implement any counter-measures, which allows for only 250 ms before the congestion occurs. However, this is nonetheless an improvement on the current process, where the counter-measures are implemented after congestion.

4.5 Results for Test Scenario 3

Scenario 3 uses the same classification model as Scenario 2. However, the training and testing process is slightly different. The same *input seed* lengths as in the previous scenarios are used, however, the model is both trained and tested on only *detected* data. We thus test the ability of the model to predict congestion without forecasting. To reduce bias during training, we employ a k-fold cross-validation [42] scheme for training and testing. The k-fold cross validation scheme splits the full dataset into *k* groups, or folds. Each fold is used as test set once, while the remaining *k-1* folds are used for training. This process is performed *k* times, such that all folds have been used for testing, and omitted from training, one time. The final result is the average score of all test.

In Scenario 3 the model is tested in two separate cases. Case 3A considers the same HGs as the previous scenarios, as well as an additional HG; HG 4. HG 4 consists of all three subgroups, i.e 10k, 20k, and 30k. In Case 3B we construct a new dataset where each data sample is a traffic pattern based on a random number of active devices drawn from an uniform distribution in the range [10000, 30000]. With this approach, we will have a dataset with a much larger state space than previously considered, and we test on a case more similar to real-life conditions. The model is still trained as a binary classification task where traffic generated from less than 20k devices do not lead to congestion, while traffic generated from more than 20k devices do lead to congestion. As mentioned in Chapter 3.3, we consider traffic from $\leq$ 20k active devices to not cause congestion, as the resulting collisions are insignificant due to maximum number of retransmissions. Traffic from more than 20k devices will always cause congestion.

We ensure equal number of data samples from each class in both cases. In Case 3A the dataset has a total of 100 samples for each test. As the state space is so small, many of the samples will be identical, and we assume 100 samples is enough to represent the full state space properly. In Case 3B we use a dataset with a total of 2000 samples. This increase is due to the much larger state space, where 100 samples is too few to fully represent the state space. The cross-validation scheme uses a k-value of 5, which is equivalent to using 80% of the dataset for training, and 20% for testing.

	Case 3A				Case 3B
Seed Length	HG 1	HG 2	HG 3	HG4	
100	100%	100%	100%	100%	98%
200	100%	100%	100%	100%	98.4%
300	100%	100%	100%	99%	98.6%
400	100%	100%	100%	100%	97.9%

Table 4.10: Accuracy on Scenario 3

We observe that the model achieves a very high accuracy on our test scenario when only using the detected traffic available at the eNB. We note that our proposed model is more precise when using the non-forecasted traffic data compared to forecasted traffic. On Case 3A, the model achieves >99% in all test condtions. On the much more general case, Case 3B, the model is still able to achieve $>$ 97% accuracy.

However, when using only the detected traffic data on the classification model, we are restricted to a set window size when classifying. For example, when the model is trained on 100 slots of detected traffic, the eNB has to use this length at all times, and would then not be able to classify congestion with a shorter, or longer, window size. Additionally, our scenarios only consider traffic during the bursty window, which suggests that the eNB first has to know if the mMTC traffic is bursty before the model is able to achieve the results shown in Table 4.10.

Nonetheless, the results achieved suggest that using ML, the eNB is able to predict congestion to a high degree, which will in turn make other improvement schemes for reducing congestion more effective.

4.6 Summary of Previous Chapters and Results

In Chapter 3 we propose a framework for solving the objectives stated in the introduction chapter. As observed in an extensive literature review, it is clear that little work has been done on the topic of traffic prediction for mMTC in OFDMA based networks, and ML based solutions are seldom proposed when researching mMTC improvements. ML has the desired attribute of inferring patterns given little information, and rivals humans on many tasks. The main trait of ML is the ability to generalize with vast amounts of data, where humans often fail. These factors motivate us to approach the thesis objective using ML techniques. In Chapter 3, an LSTM based solution is proposed for traffic forecasting, as LSTM architectures often perform more efficiently when working with time-series data compared to other ML architectures. A feed-forward NN model is proposed as a solution to the congestion classification objective. The objective of forecasting traffic arrivals and predicting congestion in a OFDMA based network cell is a challenging task, as the principal component responsible for the congestion is the number of active devices transmitting data at any given time. As this number is estimated to grow towards hundreds of thousands in the next few years, it is a challenge to generalize the solution well for this massive state space. Therefore, we consider only a small discrete state space in this thesis, with several cases for different scenarios. With this approach, we can test the ability of our model on the simpler cases proposed, and validate the effectiveness of our framework before continuing with a more general scenario.

The results in Chapter 4 suggests that the algorithm for traffic forecasting proposed in Chapter 3 is able to learn the general trend of the traffic, and the proposed models are capable of accurately predicting traffic level up to a certain time slot range. However, for longer time slot ranges, the traffic prediction model's accuracy decreases. Using a different ML architecture which can capture more information from the limited feature space might be helpful. Newer research have shown promising results in combining LSTMs with auto-encoder type networks [43], which are helpful in reducing the feature space and only preserve the most important data features. For our classification task, we are able to precisely predict congestion when using both the forecasted and non-forecasted traffic data, with the model showing very promising results in a general case with non-forecasted traffic.

Chapter 5

Factors Relevant to Numerical Results and Further Discussions

This chapter will discuss factors and other elements relevant to the results presented in Chapter 4. Some additional topics not significant to the numerical results are also discussed.

5.1 Data Normalization and Standardization

A common technique when working with time-series data is normalization, or standardization. Normalizing data often refers to the process of scaling the values of the dataset to the range [0,1]. This has shown to improve convergence time when doing gradient descent optimization for ML algorithms [41]. However, for the experiments in this thesis, using normalization for traffic prediction did not provide any better results, and in most cases yielded worse results when predicting. One explanation for this outcome could be the use of the identity function as the activation function in the output layer, which does not bound the output to the range [0,1]. Using a different activation function which bounds the output value to the range [0,1] could show to improve training when used in combination with normalization. Standardization and normalization are often used interchangeably, however in this thesis we regard normalization as the process of min-max feature scaling. We regard standardization as normalizing using the standard score, which yields values with mean 0 and standard deviation of 1.

5.2 Machine Learning Parameters

When training an ML architecture such as an LSTM, there are several parameters that affect the performance of the model. Most of the parameters, such as the number of hidden layers and hidden neurons, affect the complexity of the network and are used for more drastic changes in performance. Other parameters, such as the learning rate, optimization function, activation functions, etc. are useful for optimizing the training process, but does not affect the model as much as the number of neurons. In this section, we explain how these parameters are configured in our models, and how they affect the result.

5.2.1 Activation Functions

The activation function is mostly used to introduce non-linearity in the model, and ensure the model is able to handle complex data. In the output layer however, the activation function is used to ensure the values are within the correct range. For the traffic prediction model, the activation functions used are the default Pytorch functions for an LSTM, which is the hyperbolic tangent function: $f(x) = tanh(x)$. This function bounds the output in the range [-1,1]. No experiments have been done using other activation functions, as the hyperbolic tangent function is known to be reliable in these scenarios. The activation function at the output level is a simple linear activation, where $f(x) = x$. As the values are not normalized, using an unbounded function is appropriate. Another activation function such as the rectified linear unit (ReLU) function is also appropriate for this architecture to introduce non-linearity. The ReLU function has the form: $f(x) = max(0, x)$. However, the unbounded attribute of ReLU makes it less ideal compared to the hyperbolic tangent function.

5.2.2 Network Complexity

The complexity of the network should often mirror the complexity of the data. However, in most scenarios it is difficult to accurately judge the complexity of a dataset. The time-series data used in this thesis has only one feature, the previous value, which implies low complexity. However, as we intend to predict the future traffic several steps ahead, we need a network able to store the information captured at the predicted steps. The data thus increase in complexity as we not only consider the exact previous step, but the full history available at the current time-step. Considering that we also intend to predict the traffic for several groups, the model does require some complexity to be able to handle the problem.

Complexity could be substantially increased by increasing the network layers and number of hidden nodes. For some problems, such as computer vision, we often assume each layer captures one feature of the image, such as angles, curves, shapes, colors, etc. As our dataset only has a few features to extract, we assume a shallow network is appropriate. For the number of hidden nodes we assume an upper bound of 2000, as this is the window size we operate with during training. Before deciding on the specific set of hidden nodes and layers, a trial-and-error process is used to eliminate the worst combinations of nodes and layers. Appendix A.1

includes additional figures illustrating the effect certain complexities has on the model and its results.

5.3 Multi-step Forecasting and Loss Calculation

A general problem of recursive multi-step prediction is the issues with propagating errors for each subsequent prediction. When calculating almost 2000 steps ahead, the predictions become unstable and inaccurate in the end. In this thesis, the multi-step prediction scheme is based on predicting one step ahead, and using this value to calculate the next step. Different approaches to this scheme, such as where the model predicts >1 steps ahead, might produce different results. These approaches uses an ML model with several output nodes, and uses the output values to calculate the next sequence of slots. However, when training prediction for multiple steps ahead simultaneously, the model is required to always predict this number of steps ahead. The proposed solution uses one-step prediction for traffic forecasting as this is more dynamic, and easier to implement with non-static number of slots to predict.

The way of training the model also influences the loss function. In this thesis, the loss is calculated for the next predicted value, and not the forecasted values afterwards. This causes the model to optimize for one-step prediction, and might cause issues when predicting multiple steps ahead. A consequence of this is that the predictions can fluctuate a lot during training, and more training might not be better. Some examples of this behaviour are presented in Appendix A.2.

One solution could be to measure the error several times per training phase, and do several rounds of backpropagation. Another solution could be to train the model to predict several time-steps simultaneously by assuming that the network always predicts discrete sequences each time, e.g. always predicts 10 steps ahead, and perform recursive multi-step prediction with these 10 values for the next 10 values.

5.4 Addition of Normal Traffic

In this thesis, only bursty traffic is considered. However, in a real-life scenario, the network traffic would consist of a combination of normal and bursty traffic. The normal traffic condition is described by 3GPP as following an uniform distribution [15] given a number of active devices. This condition is composed of traffic with mostly deterministic arrival patterns, contrary to the bursty traffic, which is primarily event-driven and stochastic. Combining normal and bursty traffic in the network will have an effect on the number of arrivals per slot, and may lead to different arrival patterns than considered in this thesis. However, we assume in this thesis that the number of active devices considered during the normal and the bursty traffic conditions are independent, and thus has no effect on our experiments. As they are independent, we assume that adding normal traffic, e.g. 2 additional arrivals per slot, for all groups would be equivalent to 0 additional arrivals per slot, when considering our objectives. As a result, we have decided not to consider bursty + normal traffic as a separate scenario, as it is implicitly included in our study.

5.5 Number of Samples in Dataset

Another topic of discussion is the number of samples during training and testing. When creating the dataset for Scenario 2, the data is based on predicted traffic from Case 1B. However, as there are little to no random factors in the model during prediction, the predicted traffic will be mostly identical for each input seed, given that the same model weights and parameters are used. As a result of this, we have to either use several different models when predicting traffic to ensure some variance in the dataset, or be content with few samples. During testing of Scenario 1, we observed that there are few instances of a model being able to predict the traffic with enough accuracy to be suitable, and so we have few models to use for data generation.

5.6 State Space

As listed in Table 4.1, the state space is mostly restricted throughout this thesis. As there is little work done previously on prediction for mMTC traffic, it is often useful to start with a smaller state space before generalizing to larger domains. We observe in our test cases that the restricted state space is still a challenge for the traffic prediction model discussed in Chapter 4.3.2. However, for the classification model the state space used is trivial.

5.7 Further Discussions

This section discusses some topics which are not significant to the numerical results.

5.7.1 Processing Time for Traffic Prediction

With the configurations used in our simulations of the bursty traffic, the time between each access opportunity is 5 ms, and the full bursty window is 10 seconds. Assuming congestion starts occurring around the 450th slot, we have a limited time-window to perform all our calculations.

The full process from reading data to provide instructions should ideally take less than 5 ms, while the upper bound would be 2.25 seconds, i.e. the time until the 450th slot. The earlier the model is able to process the information, the better the model is. As mentioned in Chapter 4.4, the classification model requires a input length of around 400 slots to achieve precise results, which allows very little time to process the information and provide instructions.

In a hypothetical deployed scenario, the training phase would be performed before the model is deployed to the eNB. This part is the most time-consuming, and as it is not trained in an online fashion, we do not have to consider this time as a factor. The process of evaluating the data and forecasting the traffic would take place at the eNB and would have some time restrictions. The time spent on evaluating the traffic, and forecasting new arrivals, should be considered when assessing the applicability of a framework such as proposed in this thesis. The processing capabilities of eNB is powerful enough to handle the data and models considered in this thesis, however, the computing speed of an eNB on this framework is not tested.

5.7.2 A Potential Algorithm for Preamble Allocation at the eNB

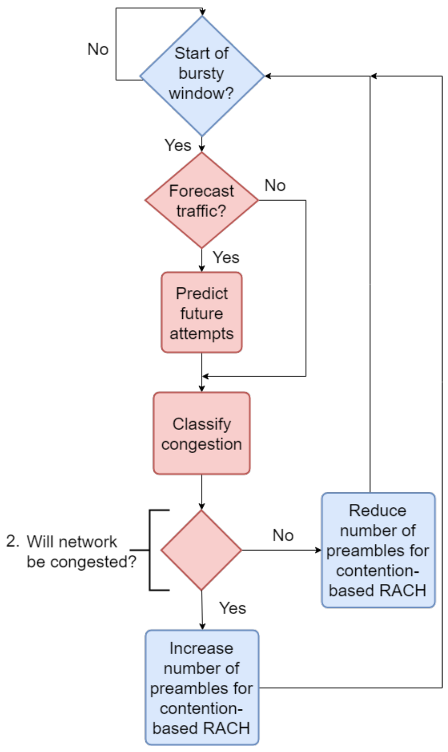

No
Start of bursty window?
Yes
Forecast traffic?
No
Yes
Predict future attempts
Classify congestion
2. Will network be congested?
No
Reduce number of preambles for contention-based RACH
Yes
Increase number of preambles for contention-based RACH

Figure 5.1: Proposed implementation of framework at eNB.

This thesis assumes in all test scenarios that the traffic condition is bursty. Therefore a separate algorithm which decides whether or not the traffic is bursty is required to replicate the results achieved in Chapter 4. If an algorithm is able to predict bursty conditions as soon as they occur, the proposed solutions studied in this thesis can be useful to predict congestion.

Additionally, algorithms to reduce or avoid congestion are also not considered in this thesis, and are a necessary next-step in the system process. The various preamble resource allocation algorithms are not necessarily as effective for any number of active devices, where some might be more effective for larger traffic volumes, and others more effective with less volume. By predicting traffic arrival, and the volume thereof, the framework could dynamically choose the best improvement based on the traffic condition.

Figure 5.1 shows the proposed framework as well as a simple algorithm for reducing congestion based on increasing the number of preambles available for contention based RACH.

Chapter 6

Conclusions and Future Work

According to the results obtained in the two scenarios, we are able to come to a conclusion regarding the objectives stated in Chapter 1. This chapter presents the conclusion for each objective, as well as an overall summary of the thesis work. Additionally, some topics for future research are presented.

6.1 Conclusions

The objectives in this thesis focus on using detected traffic arrivals to predict congestion in an OFDMA based mMTC network. Based on our observation that very few ML based approaches exists for mMTC traffic prediction, we propose an ML based solution scheme to resolve these issues. ML has shown to produce excellent results on time-series data, and is commonly used to solve these types of problems. However, as our data is univariate, we have a challenging task in forecasting arrivals using ML. To ensure the best foundation for our solution, we first analyze simulated traffic patterns to learn more about the different traffic conditions. Based on the information gathered, we train an LSTM network in one-step prediction, and use the predictions in a recursive multi-step prediction scheme for traffic forecasting. We test the forecasted data on an ML based classifier, and compare the results with the non-forecasted traffic data.

Our first objective is to predict traffic arrival using only previous detected arrivals, which is the only information available to the eNB. The detected arrivals are directly dependent to the number of active devices in a cell. However, because of the randomness introduced by the preamble selection scheme, and potential interference, it is difficult to calculate the number of active devices given the detected arrivals. To solve our problem, we utilize ML techniques to predict the number of active devices in a cell. But, because the range of possible numbers is so large, we first simplify our task by assuming a discrete number of possible active devices: 10k, 20k, and 30k. We arrange the three possibilities in groups of two, for a total of three grouping arrangements. We train on two discrete classes of active devices at a time, and our LSTM based solution framework is able to distinguish and predict the future traffic in certain cases, but unable to reach an acceptable accuracy for the groups with similar arrival patterns. The traffic generated by 20k and 30k active devices follow a very similar arrival pattern, and the model is unable to properly distinguish these two groups and predict accordingly.

Our second objective is predicting congestion in the network, and here we propose a simple feed-forward NN to classify the traffic. We model this as a binary classification task, where the traffic is classified as either leading to congestion or not leading to congestion. We train the model on simulated traffic arrivals following distributions provided by 3GPP, and test on both the forecasted traffic and original detected traffic. Using the original traffic data we are able to achieve >97.5% accuracy in all traffic conditions. With the forecasted traffic, the model is able to achieve satisfactory accuracy by using input seeds of length 400, while for the other input seed lengths the model does not achieve a satisfactory accuracy. We also note that the model is able to generalize reasonably well, and the model should perform adequately for less restricted scenarios than considered in this thesis.

We conclude that by classifying already available traffic, we can precisely predict the occurrence of congestion in the bursty window. The process of forecasting traffic arrivals does not improve the prediction, but we note that the proposed LSTM model is able to forecast traffic following the same trend as the real traffic, and we assume that using better techniques will produce more convincing results. We observe that the algorithms proposed here have shown potential, and expect them to be adaptable for more general traffic scenarios as well.

6.2 Thesis Contributions

This section lists the main contributions made by this thesis work.

- A framework for predicting future traffic arrivals and predicting congestion in bursty mMTC traffic, which is purely based on the preambles received as the eNB, has been proposed.
- Arrivals patterns for bursty mMTC traffic have been analyzed, and the the access success probabilities for 10k, 20k, and 30k active devices have been calculated.
- An ML model for forecasting traffic arrival in mMTC network has been designed and verified.
- An ML model for classifying congestion status for traffic in mMTC network during bursty conditions has been designed and verified.

- The framework has been implemented and tested for several scenarios.

6.3 Future Work

Some potential topics and directions for future research are presented here.

- The main advancement to this thesis work would be to test the proposed solutions on a dataset based on real-life traffic traces. This will provide insight into how the traffic is affected by normal traffic condition in addition to the bursty traffic, as well as how the traffic is affected by channel impairments.
- We only consider certain discrete options, such as 10k, 20k, and 30k, in regards to the number of active devices in a cell in this thesis. In future work it would be necessary to generalize the environment more than we have done in this work. This might also require a different approach to the solution than proposed here.
- The chosen loss function calculates the MSE based on one-step prediction although our tasks are based on multi-step forecasting. Further research should be done on the topic of calculating loss for multiple time-steps.

Bibliography

[1] Reuters, *Global Smart cities Market Size and Trends* [Online]. Available: https://www.reuters.com/brandfeatures/venture-capital/article?id=30881 Accessed: 23. Mar. 2019.

[2] S.N. Deepa, N. Arulmozhi, B. Gobu, P. Kanimozhi, S. Jaikumar, and A. A. V. Tangaradjou, ”Adaptive Regularized ELM and Improved VMD Method for Multi-step ahead Electricity Price Forecasting,” *IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 1255-1260, 2018.

[3] S. Ali, W. Saad, and N. Rajatheva, “A Directed Information Learning Framework for Event-Driven M2M Traffic Prediction,” *IEEE Communications Letters*, vol. 22, no. 11, pp. 2378-2381, Nov. 2018.

[4] T. N. Weerasinghe, I. A. M. Balapuwaduge, and F. Y. Li, “Supervised Learning based Arrival Prediction and Dynamic Preamble Allocation for Bursty Traffic,” *IEEE International Conference on Computer Communications (INFOCOM) Workshops*, Apr. 2019.

[5] A. Sivanathan et al., “Characterizing and Classifying IoT Traffic in Smart Cities and Campuses,” *IEEE Conference on Computer Communications (INFOCOM) Workshops)*, pp. 559-564, 2017.

[6] S. K. Sharma and X. Wang, “Towards Massive Machine Type Communications in Ultra-Dense Cellular IoT Networks: Current Issues and Machine Learning-Assisted Solutions,” *IEEE Communications Surveys —& Tutorials*, May 2019

[7] M. S. Ali, E. Hossain, and D. I. Kim, “LTE/LTE-A Random Access for Massive Machine-Type Communications in Smart Cities,” *IEEE Communications Magazine*, vol. 55, no. 1, pp. 76-83, Jan. 2017.

[8] R. Cheng, C. Wei, S. Tsao, and F. Ren, “RACH Collision Probability for Machine-Type Communications,” *IEEE Vehicular Technology Conference (VTC Spring)*, pp. 1-5, 2012.

[9] 3GPP TR36.300, “Evolved Universal Terrestrial Radio Access (E-UTRA) and Evolved Universal Terrestrial Radio Access Network (E-UTRAN),” v15.4.0, Jan. 2019.

[10] 3GPP TR38.913, “Study on Scenarios and Requirements for Next Generation Access Technologies,” v15.0.0, Jul. 2018.

[11] 3GPP, “LTE,” [Online]. Available: http://www.3gpp.org/technologies/keywords-acronyms/98-lte Accessed: 24. Mar. 2019.

[12] 3GPP TR36.913, “Requirements for Further Advancements for Evolved Universal Terrestrial Radio Access (E-UTRA),” v15.0.0, Jul. 2018.

[13] 3GPP, “LTE-Advanced,” [Online]. Available: http://www.3gpp.org/technologies/keywords-acronyms/97-lte-advanced 2013. Accessed: 24. Mar. 2019.

[14] 3GPP TR32.826, “Study on Energy Savings Management (ESM),” v1.0.0, Dec. 2009.

[15] 3GPP TR37.868, “RAN Improvements for Machine-type Communications,” v11.0.0, Oct. 2011.

[16] 3GPP TR21.915, “Release description; Release 15,” v0.6.0, Feb. 2019.

[17] 3GPP TR38.331, “NR; Protocol specification,” v15.4.0, Jan. 2019.

[18] W. Held, “5G Enabled by Massive Capacity, Connectivity,” Nokia, [Online]. Available: https://www.nokia.com/blog/5g-enabled-massive-capacity-connectivity/ Accessed: Mar. 2019.

[19] A. D. Zayas and P. Merino, “The 3GPP NB-IoT System Architecture for the Internet of Things,” *IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 277-282, May 2017.

[20] C. Bockelmann et al., “Massive Machine-type Communications in 5G: Physical and MAC-layer Solutions,” *IEEE Communications Magazine*, vol. 54, no. 9, pp. 59-65, Sept. 2016.

[21] H. Shariatmadari et al., “Machine-type Communications: Current Status and Future Perspectives Toward 5G Systems,” *IEEE Communications Magazine*, vol. 53, no. 9, pp. 10-17, Sept. 2015.

[22] M. Hyder and K. Mahata, “Zadoff–Chu Sequence Design for Random Access Initial Uplink Synchronization in LTE-Like Systems,” *IEEE Trans. Wireless. Comm*, Jan. 2017.

[23] A. Benjebbour, “An Overview of Non-orthogonal Multiple Access,” *ZTE Communications*, vol. 15, No. S1, Jun. 2017.

[24] MYMO Wireless, “Preamble Detection and Timing Advance Estimation for multi-UE in 3GPP LTE,” [Online]. Available: https://www.mymowireless.com/wp-content/uploads/2017/02/White-Paper-PRACH-Preamble-Detection-and-Timing-Advance-Estimation-for-....pdf Accessed: 19. Mar. 2019.

[25] O. Yang and W. Heinzelman, “Modeling and Performance Analysis for Duty-Cycled MAC Protocols with Applications to S-MAC and X-MAC,” *IEEE Transactions on Mobile Computing*, vol. 11, no. 6, pp. 905-921, Jun. 2012.

[26] J. Oller, I. Demirkol, J. Casademont, J. Paradells, G. U. Gamm, and L. Reindl, “Has Time Come to Switch From Duty-Cycled MAC Protocols to Wake-Up Radio for Wireless Sensor Networks?,” *IEEE/ACM Transactions on Networking*, vol. 24, no. 2, pp. 674-687, Apr. 2016.

[27] L. Wilhelmsson and D.Sundman, “Wake-Up Radio - A key component of IoT?,” Ericsson, [Online]. Available: https://www.ericsson.com/en/blog/2017/12/wake-up-radio–a-key-component-of-iot. Dec. 2018.

[28] A. Froytlog et al., ”Ultra-low Power Wake-up Radio for 5G IoT,” *IEEE Communications Magazine*, vol. 57, no. 3, pp. 111-117, March 2019.

[29] P. Louridas and C. Ebert, “Machine Learning,” *IEEE Software*, vol. 33, no. 5, pp. 110-115, Sept. 2016.

[30] W. J. Zhang, G. Yang, Y. Lin, C. Ji, and M. M. Gupta, “On Definition of Deep Learning,” *World Automation Congress (WAC)*, pp. 1-5, Jun. 2018

[31] X. Du, Y. Cai, S. Wang, and L. Zhang, “Overview of Deep Learning,” *Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pp. 159-164, 2016.

[32] C. Olah, “Understanding LSTM Networks,” [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ Accessed: 27. Mar. 2019.

[33] A. Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks,” [Online]. Available: https://karpathy.github.io/2015/05/21/rnn-effectiveness/ Accessed: Mar. 2019.

[34] J. Moon and Y. Lim, “A Reinforcement Learning Approach to Access Management in Wireless Cellular Networks,” *Wireless Communications and Mobile Computing*, pp. 1-7, 2017.

[35] L. Tello-Oquendo, D. Pacheco-Paramo, V. Pla, and J. Martinez-Bauset, “Reinforcement Learning-Based ACB in LTE-A Networks for Handling Massive M2M and H2H Communications,” *IEEE International Conference on Communications (ICC)*, pp. 1-7, 2018.

[36] D. Magrin, C. Pielli, C. Stefanovic, and M. Zorzi, “Enabling LTE RACH Collision Multiplicity Detection via Machine Learning,” [Online]. Available: https://arxiv.org/abs/1805.11482 2018. Accessed: 5. Apr. 2019

[37] W. H. Press and S. A. Teukolsky, “Savitzky-Golay Smoothing Filters,” *Computers in Physics*, vol. 4, no. 6, pp. 669-672, 1990.

[38] S. Fadnavis, “A Generalization of the Birthday Problem and the Chromatic Polynomial,” [Online]. Available: https://ui.adsabs.harvard.edu/abs/2011arXiv1105.0698F. Accessed: 15. May. 2019.

[39] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *ICLR*, 2015.

[40] S. Gomar, M. Mirhassani and M. Ahmadi, “Precise Digital Implementations of Hyperbolic Tanh and Sigmoid Function,” *Asilomar Conference on Signals, Systems and Computers*, pp. 1586-1589, 2016.

[41] S. Ruder, “An Overview of Gradient Descent Optimization Algorithms,” [Online]. Available: http://arxiv.org/abs/1609.04747 Accessed: 12. Mar. 2019

[42] S. Yadav and S. Shukla, “Analysis of k-Fold Cross-Validation over Hold-Out Validation on Colossal Datasets for Quality Classification,” *IEEE 6th International Conference on Advanced Computing (IACC)*, pp. 78-83. 2016.

[43] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi, “Sequence-to-Sequence Prediction of Vehicle Trajectory via LSTM Encoder-Decoder Architecture,” *IEEE Intelligent Vehicles Symposium (IV)*, 2018.

Appendices

Appendix A

Additional Simulation Results

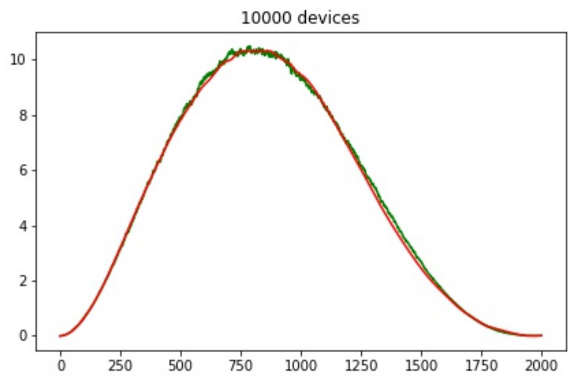
In the following appendix, supplementary figures are included to illustrate the volatile nature of training an LSTM on time-series data, as well as the effect different parameter values have on an ML model.

The following figures are a result of the trial-and-error procedure in choosing the best parameters for the traffic prediction model.

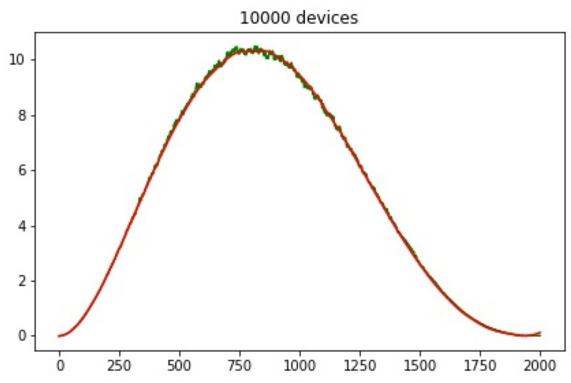
A.1 Effect of Using Various Combinations of Complexity on the Traffic Prediction Model

The following figures compare forecasted traffic based on models trained with different complexities. The models are based on the homogeneous case, as the best results from this test case were used to initialize the models in the heterogeneous case via transfer learning. The figures show results for 10k, 20k, and 30k, using values for hidden nodes equal to 500, 1000, or 1500, and values for hidden layers equal to 2 or 3.

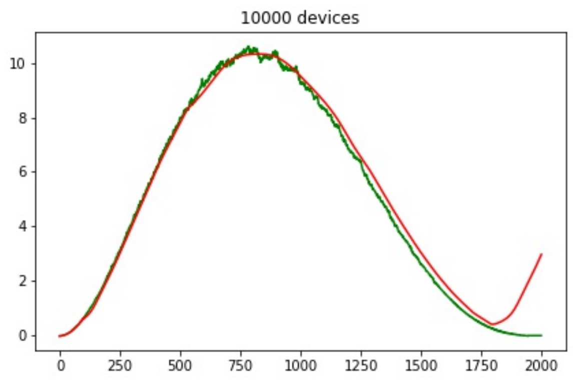
Forecasting traffic for 10k active devices



(a) 500 hidden nodes and 2 hidden layers

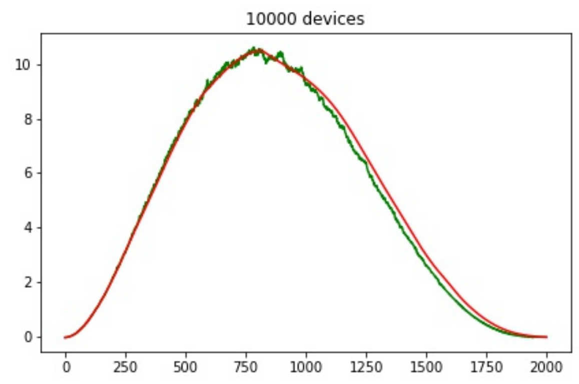


(b) 1000 hidden nodes and 2 hidden layers

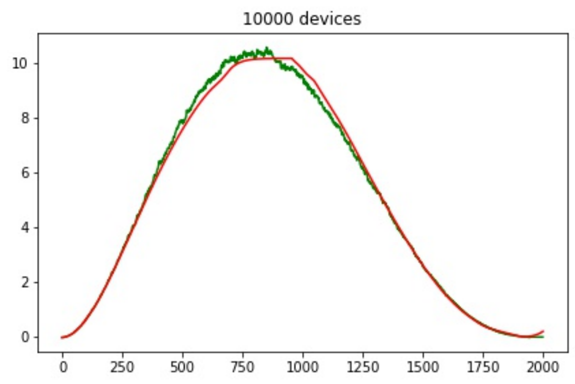


(c) 1500 hidden nodes and 2 hidden layers

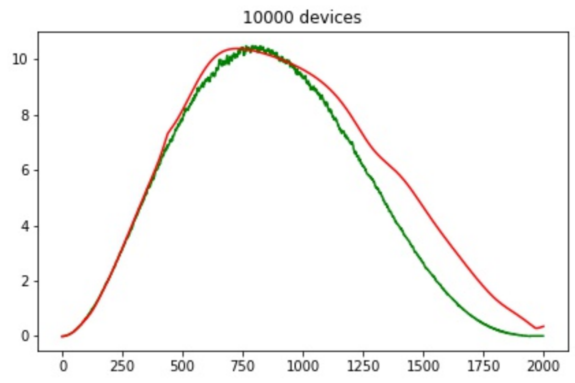
Figure A.1: Effect of Different Number of Hidden Nodes in 2-layer LSTM for Forecasting Traffic based on 10k Active Devices.



(a) 500 hidden nodes and 3 hidden layers



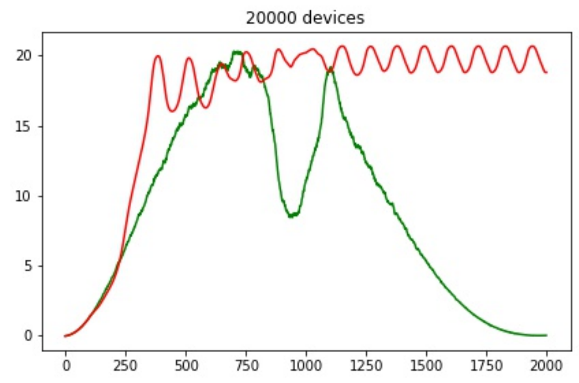
(b) 1000 hidden nodes and 3 hidden layers



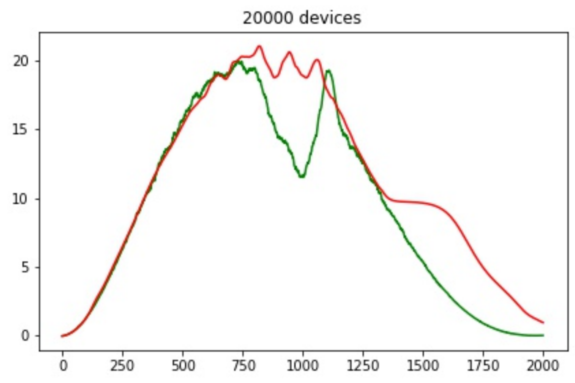
(c) 1500 hidden nodes and 3 hidden layers

Figure A.2: Effect of Different Number of Hidden Nodes in 3-layer LSTM for Forecasting Traffic based on 10k Active Devices.

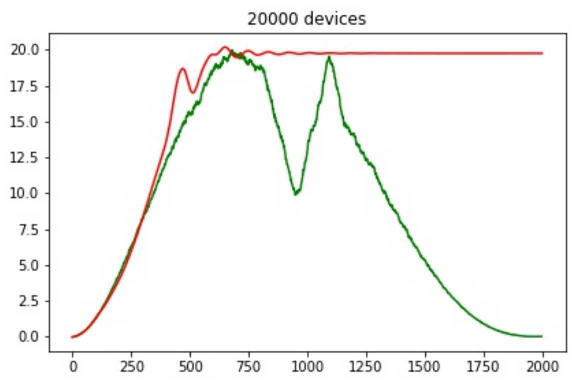
Forecasting traffic for 20k active devices



(a) 500 hidden nodes and 2 hidden layers

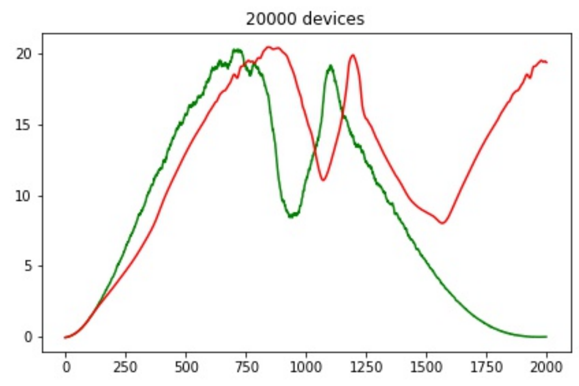


(b) 1000 hidden nodes and 2 hidden layers

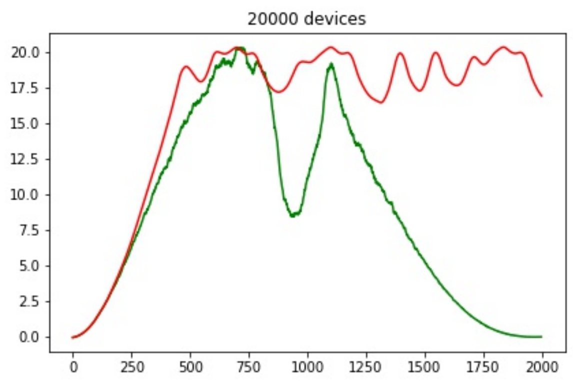


(c) 1500 hidden nodes and 2 hidden layers

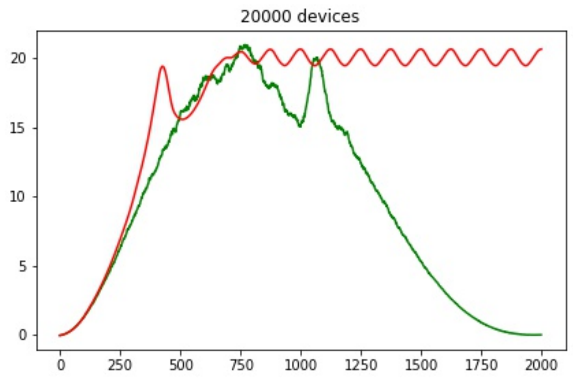
Figure A.3: Effect of Different Number of Hidden Nodes in 2-layer LSTM for Forecasting Traffic based on 20k Active Devices.



(a) 500 hidden nodes and 3 hidden layers



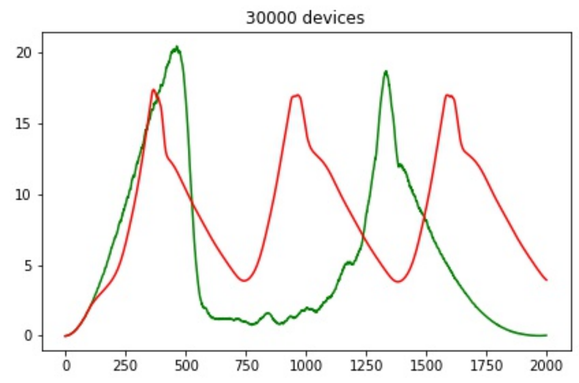
(b) 1000 hidden nodes and 3 hidden layers



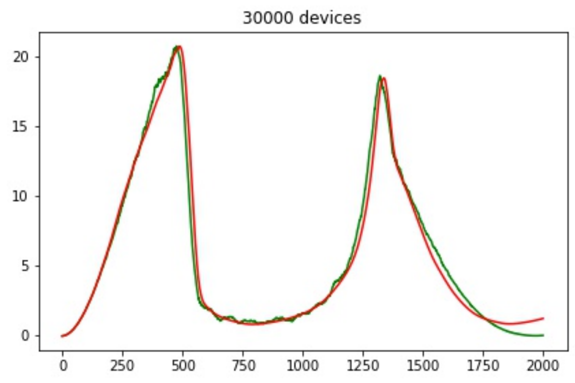
(c) 1500 hidden nodes and 3 hidden layers

Figure A.4: Effect of Different Number of Hidden Nodes in 3-layer LSTM for Forecasting Traffic based on 20k Active Devices.

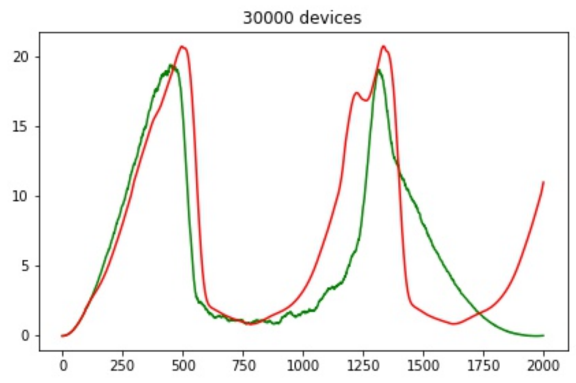
Forecasting traffic for 30k active devices



(a) 500 hidden nodes and 2 hidden layers

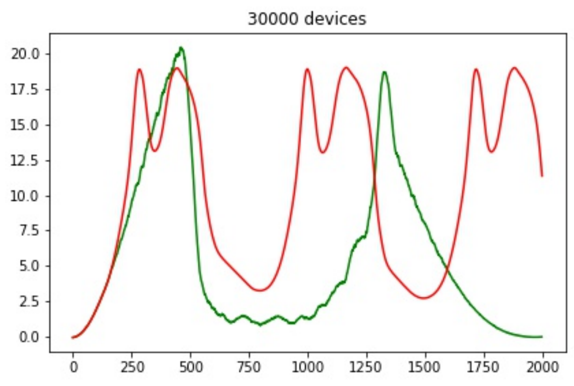


(b) 1100 hidden nodes and 2 hidden layers

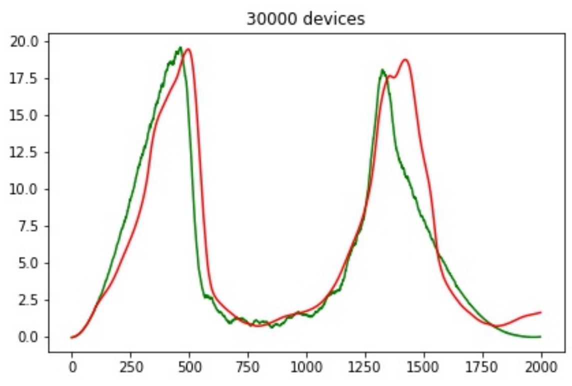


(c) 1500 hidden nodes and 2 hidden layers

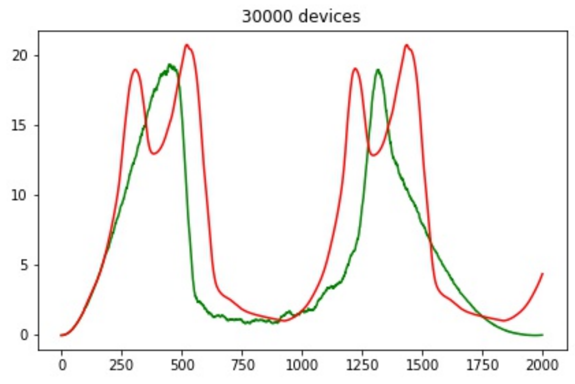
Figure A.5: Effect of Different Number of Hidden Nodes in 2-layer LSTM for Forecasting Traffic based on 30k Active Devices.



(a) 500 hidden nodes and 3 hidden layers



(b) 1000 hidden nodes and 3 hidden layers

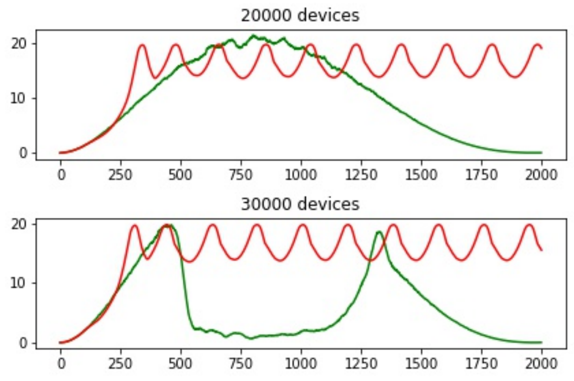


(c) 1500 hidden nodes and 3 hidden layers

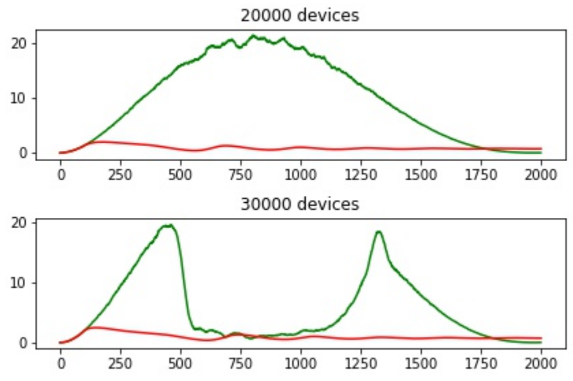
Figure A.6: Effect of Different Number of Hidden Nodes in 3-layer LSTM for Forecasting Traffic based on 30k Active Devices.

We observe in this section that most combinations of nodes and layers are able to forecast the traffic to some extent, but line between too little complexity and too much is a very fine line. By increasing the complexity too much, the model requires much more training to be optimize all the nodes properly. With less complex models, the model might not have enough nodes to fit the approximated functions. With few nodes, we observe in Figures A.5(a) and A.6(a) that the predicted traffic becomes cyclical after a short period, which might indicate that the model is unable to learn longer predictions with few nodes. For Figure A.5(a), the cycle begins after around 750 slots, which implies it is unable to learn complex time-series of more than 750 slots with only 500 nodes. The same pattern emerges for Figure A.6(a), which implies that a deeper network, more layers, does not improve long term predictions.

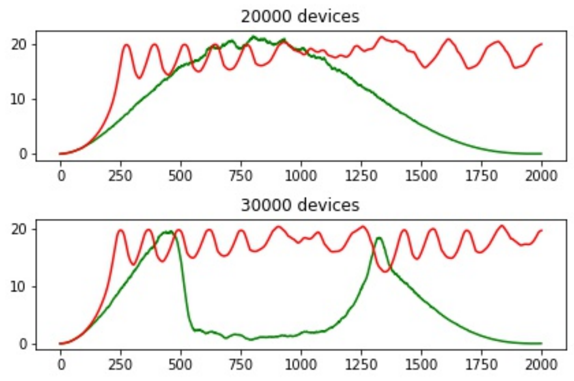
A.2 Effect of Additional Epochs During Training



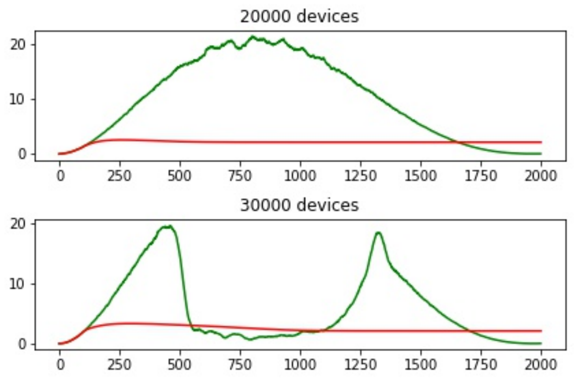
(a) Forecasted traffic after 3000 epochs



(b) Forecasted traffic after 3200 epochs



(c) Forecasted traffic after 3400 epochs



(d) Forecasted traffic after 3600 epochs

Figure A.7: Effect of Additional Training on Traffic Forecast for HG3.

Figure A.7 illustrates how the forecasted traffic does not converge to the optimal solution after training for additional epochs. Even though the MSE might decrease between each visualization, the forecasted traffic switches between 20 arrivals per slot, and 5 arrivals per slot. The forecasted traffic was in this trial visualized every 200th epoch. In each sub-figure, the traffic is forecasted for 20k and 30k active devices, as this is the groups the model was trained on.

Appendix B

Python/Pytorch Code Samples

The following section includes the code used to implement the algorithms discussed in Chapter 4. The Python code is written using Python version 3.6.7. The Pytorch library is used for ML programming, using Pytorch version 1.0 and CUDA version 10 for GPU acceleration

B.1 Algorithm 4.1

```
from pathlib import Path
from scipy import special
from scipy import integrate
from scipy import signal
import numpy as np

class ue():
    def __init__(self):
        self.transmissions = 0
        self.preamble = 0
        self.packet_size = 0
        self.backoff_counter = 0
        self.group = 0 # group 0 = A, 1 = B

class GenerateTraffic():

    ## Define simulation parameters
    def __init__(self):
        self.frame_size = 0.005
        self.T = 10            #total traffic window of 10 seconds
        self.num_devices = 30000
        self.preambleTransMax = 10 #max number of retransmissions as
            specified by 3GPP
        self.backoffWindowMax = 20 # units in milliseconds.
        self.bursty_window_size = int(self.T/self.frame_size)


    def beta_pdf(self, t):
        alpha = 3
        beta = 4
        beta_function = special.beta(alpha, beta)
        return (t**(alpha-1)*(self.T-t)**(beta-1))/(self.T**(alpha+beta-1)*
            beta_function)


    ## Function to generate traffic following uniform distribution for
        normal traffic condition
    def create_uniform_traffic(self, n_d=None, T=60,):
        #set default time T to 60s
        if(n_d == None): n_d = self.num_devices
        slots = round(T / self.frame_size)
        devices_per_slot = n_d / slots
        access_intensity = np.array([devices_per_slot for i in range(slots)
            ])

        uniform_traffic = np.zeros((slots))
        tracker = 0
        for i, v in enumerate(access_intensity):
            v += tracker
            tracker = 0
            uniform_traffic[i] = np.floor(v)
            tracker = tracker + (v - np.floor(v))
        return uniform_traffic, access_intensity

    ## Function to generate traffic following the Access Intensity function
         based on the 3GPP Report.
    def create_bursty_traffic(self, n_d=None):
        if(n_d == None): n_d = self.num_devices
        a_t = [] #each index keeps a float with the "access intensity" at
            that time t

        ## Access Intensity Function
        ## Calculate arrivals per time-slot
        for t in np.linspace(0,self.T-self.frame_size, self.
            bursty_window_size):
            approx_integral = integrate.quad(lambda t: self.beta_pdf(t), t,
                 t+self.frame_size)[0]
            access_intensity = n_d * approx_integral
            a_t.append(access_intensity)

        ## Aggregate bursty traffic to 5ms ra slots
        initial_arrivals = np.zeros(self.bursty_window_size) #devices
            attempting in each RA slot. Does not include retransmissions
        tracker = 0
        for i, v in enumerate(a_t):
            v += tracker
            tracker = 0
            initial_arrivals[i] = np.floor(v)
            tracker = tracker + (v - np.floor(v))

        return initial_arrivals, a_t


    ## Function to simiulate collisions and retransmissions
    def simulate_bursty_traffic_arrival(self, initial_arrivals, groups=
        False, backoff_bool=True):
        window_size = initial_arrivals.shape[0]
        ues_per_slot = np.zeros(window_size)
        ues = []
        group_treshold = 0.85
        success_per_slot = np.zeros(window_size)
        training_data = [] #X = registered preambles at eNB per RA slot as
            one-hot encoded vector. Y = actual attempts against eNB
        for slot, new_arrivals in enumerate(initial_arrivals):
            successful_preambles = np.zeros(54)
            unsuccessful_preambles = []
            preamble_counter = {k:0 for k in range(54)} #track how often a
                preamble is used

            # create n new devices each RA slot
            for i in range(int(new_arrivals)):
                uequipment = ue()
                ## has to be done better with actual packet size
                    distributions.
                ## this is based on uniform distribution which makes groups
                     unnecessary
                if(groups):
                    r = np.random.randint(0,100)
                    if(r > group_treshold):
                        uequipment.group = 1
                ues.append(uequipment)

            # give each UE a preamble
            for device in ues:
                # if UE is still in backoff, do not give preamble
                if(device.backoff_counter > 0):
                    device.preamble = 99
                    # continue
                else:
                    if(groups):
                        split = np.ceil(54*group_treshold)
                    else:
                        split = 54
                    if(device.group == 0):
                        preamble = np.random.randint(0, split)
                    else:
                        preamble = np.random.randint(split, 54)

                    device.preamble = preamble
                    preamble_counter[preamble] += 1
                    device.transmissions += 1

            # fill RA slot one-hot vector with correct preambles
            # checks for collisions
            for j, v in enumerate(successful_preambles):
                j = int(j)
                if(preamble_counter[j] == 1):
                    successful_preambles[j] = 1

            success_per_slot[slot] = (sum(successful_preambles))
            # check individual UE's for success.
            finished_ues = []
            ues_per_slot_counter = 0
            for device in ues:
                if(device.backoff_counter == 0):
                    ues_per_slot_counter += 1 #only count transmitting ues
                        in each slot
                    if(preamble_counter[device.preamble] == 1):
                        finished_ues.append(device)
                    elif(device.transmissions == self.preambleTransMax):
                        finished_ues.append(device)
                    else:
                        if(backoff_bool):
                            rand_backoff = np.random.randint(0, self.
                                backoffWindowMax+1)
                        else:
                            rand_backoff = 0
                        device.backoff_counter = np.ceil(rand_backoff/(self
                            .frame_size*1000))
                else:
                    device.backoff_counter -= 1

            for device in finished_ues:
                ues.remove(device)
                del(device)
```

```
            ues_per_slot[slot] = ues_per_slot_counter

        return success_per_slot, ues_per_slot


if __name__=="__main__":
    traffic_generator = GenerateTraffic()
    bursty_traffic, a_t = traffic_generator.create_bursty_traffic(n_d=n) #
        bursty window is 2k slots
    detected, attempted = traffic_generator.simulate_bursty_traffic_arrival
        (bursty_traffic, backoff_bool=True)
```

B.2 Algorithm 4.2

```
import torch
from torch import nn
from torch.autograd import Variable
from torch import optim
import torch.nn.functional as F
import alg_41_traffic_generator as alg_41
cuda = torch.device('cuda')

class FPredRNN(nn.Module):
    def __init__(self, features, batch_size, num_hidden, num_layers,
        dropout_val, track_states=False):
        super(FPredRNN, self).__init__()
        self.bs = batch_size
        self.features = features
        self.nh = num_hidden
        self.nl = num_layers
        self.track = track_states

        self.rnn1 = nn.LSTM(self.features, self.nh, self.nl) #(features,
            hidden_size, num_layers)
        self.dropout = nn.Dropout(p=dropout_val)
        self.linear = nn.Linear(self.nh, 1)

        self.init_hidden(self.bs)

    def init_hidden(self, batch_size):
        self.h0 = torch.zeros(self.nl, batch_size, self.nh, dtype=torch.
            double, device=cuda) # (num_layers * num_directions, batch_size,
             hidden_size)
        self.c0 = torch.zeros(self.nl, batch_size, self.nh, dtype=torch.
            double, device=cuda) # num_directions = 2 for bidirectional,
            else 1

    def forward(self, inputs, steps=1000, eval=False):
        predictions = []
        batch_size = inputs.size(1)
        if(self.h0.size(1) != batch_size):
            self.init_hidden(batch_size)

        outputs, (hn,cn) = self.rnn1(inputs, (self.h0, self.c0))
        out = self.dropout(outputs)
        output = self.linear(out)

        if(eval):
            eval_input = output[-1:]
            for i in range(steps):
                lstm_out, (hn,cn) = self.rnn1(eval_input, (hn, cn))
                linear_out = self.linear(lstm_out)
                predictions += [linear_out]
                eval_input = linear_out

            output = torch.stack(predictions).squeeze()
        return output


if __name__=="__main__":
    x_data = np.empty((2,traffic_generator.bursty_window_size))
    y_data = np.empty((2,1))
    for n in [10000,20000]:
        traffic_generator = alg_41.GenerateTraffic()
        bursty_traffic, a_t = traffic_generator.create_bursty_traffic(n_d=n
            )
        detected, attempted = traffic_generator.
            simulate_bursty_traffic_arrival(bursty_traffic, backoff_bool=
            True)
        smooth_x = signal.savgol_filter(detected, 97, 2)
        x_data[n] = smooth_x

    inputs = x_data[:,:1999]
    targets = x_data[:,1:2000]

    model = FPredRNN(inputs.size(2), batch_size, num_hidden, num_layers,
        dropout_val, track_states=False)
    model.double()
    model.cuda()
    model.train()
    model.init_hidden(batch_size)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=0.0001)

    #Train model
    optimizer.zero_grad()
    out = model(inputs)
    loss = criterion(out, targets)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    torch.save(model, 'trained_model.pt')

    #Test model
    seed_length = 100
    seed = inputs[:seed_length]
    out = model(seed, steps=steps, eval=True)
    test_out = torch.cat((seed.squeeze(), out))
```

B.3 Algorithm 4.3

```
import torch
from torch import nn
from torch.autograd import Variable
from torch import optim
import torch.nn.functional as F
cuda = torch.device('cuda')

import alg_41_traffic_generator as alg_41
import alg_42_prediction_model as alg_42

class ClassificationModel(torch.nn.Module):
    def __init__(self, features, classes, hidden_nodes):
        super(ClassificationModel, self).__init__()
        self.nh = hidden_nodes
        self.linear1 = nn.Linear(features, classes) # number of slots in,
            and 2 features out
        self.dropout = nn.Dropout(p=0.2)

    def forward(self, inputs):
        output = self.dropout(self.linear1(inputs))
        return output


if __name__=="__main__":
    x_data = np.empty((2,traffic_generator.bursty_window_size))
    y_data = np.empty((2,1))
```

```
labels = [0,1]
for n in [10000,20000]:
    traffic_generator = alg_41.GenerateTraffic()
    bursty_traffic , a_t = traffic_generator.create_bursty_traffic()
    detected , attempted = traffic_generator.
        simulate_bursty_traffic_arrival(bursty_traffic , backoff_bool=
        True)

    smooth_x = signal.savgol_filter(detected , 97, 2)
    x_data[n] = smooth_x
    y_data[n] = labels[j]



for distance in [100, 1000, 2000]:
    inputs = x_data[:,:distance]
    targets = y_data

    model = ClassificationModel(inputs.size(0), inputs.size(1))
    model.double()
    model.cuda()
    model.train()

    criterion = torch.nn.BCEWithLogitsLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.0001)

    ## Train model
    optimizer.zero_grad()
    y_pred = model(inputs)
    loss = criterion(y_pred, targets)
    loss.backward()
    optimizer.step()

    ## Test model
    seed_length = 100
    # Load trained model from Alg. 4.2
    forecast_model = torch.load('trained_model.pt')
    forecasted_traffic = forecast_model(x_data[:seed_length])
    predicted_class = np.argmax(model(forecasted_traffic))
    predicted_number_of_active_devices = [10000,20000][predicted_class]
```