YAŞAR UNIVERSITY

GRADUATE SCHOOL

MASTER’S THESIS

MACHINE LEARNING BASED MULTI-SCALE JOINT FORECASTING-SCHEDULING FOR THE INTERNET OF THINGS

MERT NAKIP

THESIS ADVISOR: ASSOC. PROF. VOLKAN RODOPLU

CO-ADVISOR: PROF. CÜNEYT GÜZELİŞ

ELECTRICAL AND ELECTRONICS ENGINEERING

PRESENTATION DATE: 01.07.2020

BORNOVA / İZMİR

JULY 2020

We certify that, as the jury, we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Jury Members:	**Signature:**
Assoc. Prof. Volkan RODOPLU Yaşar University	[illegible]
Prof. Cüneyt GÜZELİŞ Yaşar University	[illegible]
Prof. Mustafa SEÇMEN Yaşar University	[illegible]
Prof. Türker İNCE İzmir University of Economics	[illegible]
Assoc. Prof. Barış ATAKAN İzmir Institute of Technology	[illegible]

[illegible]

Prof. Cüneyt GÜZELİŞ

Director of the Graduate School

ABSTRACT

MACHINE LEARNING BASED MULTI-SCALE JOINT FORECASTING-SCHEDULING FOR THE INTERNET OF THINGS

NAKIP, Mert
MSc, Electrical and Electronics Engineering
Advisor: Assoc. Prof. Volkan RODOPLU
Co-Advisor: Prof. Cüneyt GÜZELİŞ
July 2020

The Massive Access Problem of the Internet of Things (IoT) is the problem of enabling the wireless access of a massive number of IoT devices to the wired infrastructure. In this thesis, we develop a Multi-Scale Algorithm (MSA) for joint forecasting-scheduling at a dedicated IoT Gateway to solve the Massive Access Problem at the Medium Access Control layer. In contrast with the current approaches to the Massive Access Problem that assume random arrivals for IoT traffic generation, our algorithm forecasts the upcoming traffic of IoT devices using a Multi-Layer Perceptron (MLP) architecture and preallocates the uplink wireless channel based on these forecasts.

In order to show that predictive resource allocation algorithms can be implemented in practice to solve the Massive Access Problem, first, we establish that the traffic generation pattern of individual IoT device can be predicted accurately with respect to the symmetric Mean Absolute Percentage Error (sMAPE) metric. To this end, we present a comparative study of the performance of Autoregressive Integrated Moving Average (ARIMA), Multi-Layer Perceptron (MLP), 1-Dimensional Convolutional Neural Network (1D CNN), and Long-Short Term Memory (LSTM) models on the problem of forecasting the traffic generation patterns of individual IoT devices in Machine-to-Machine communication. We classify IoT traffic into four classes: Fixed-Bit Periodic (FBP), Variable-Bit Periodic (VBP), Fixed-Bit Aperiodic (FBA), and Variable-Bit Aperiodic (VBA). In our simulation studies, we show that LSTM outperforms all of the other forecasting models significantly for IoT devices in the VBP class. In addition, we show that LSTM has almost the same performance in sMAPE for the FBA class as MLP and 1D CNN. While the training time per IoT device is the highest for LSTM, all of the forecasting models have reasonable training times for practical implementation.

Based on these results that demonstrate the predictability of IoT traffic generation patterns, in order to alleviate the Massive Access Problem of the IoT, we propose a Joint Forecasting-Scheduling (JFS) system to be implemented at an IoT gateway. Our JFS system forecasts the traffic generation pattern of each IoT device in its coverage area and schedules the uplink transmissions of these devices in advance in a collision-free manner. Compared with reactive solutions to the Massive Access Problem, JFS has a significant advantage in that it obviates contention, collision and handshaking. Next, we compare the performance of ARIMA, MLP and LSTM forecasting models in order to select a local-optimal forecasting model that achieves the highest network throughput for the JFS system. This comparison reveals that the optimal choice of the forecasting model for JFS depends heavily on the proportions of distinct IoT device classes that are present in the network. Our network simulations show that the JFS scales up to 1000 devices with a forecasting-scheduling window size as long as 1800 seconds, while achieving a total execution time under 1 second.

In order to scale the JFS system to support more than 1000 devices and achieve scheduling over a longer forecasting-scheduling window size, we design MSA, which operates at multiple time scales that are determined by the delay constraints of IoT applications as well as the minimum traffic generation periods of IoT devices. The multi-scale nature of our algorithm ensures scalable time and space complexity to support up to 6650 IoT devices in our simulations. We compare the throughput and energy consumption of MSA with those of Reservation-based Access Barring, Priority based on Average Load, and Enhanced Predictive Version Burst Oriented protocols, and show that MSA significantly outperforms these beyond 3000 devices. Furthermore, we show that the percentage control overhead of MSA remains less than 1.5%.

The results of this thesis pave the way to building scalable joint forecasting-scheduling systems to handle a massive number of IoT devices at IoT Gateways.

Keywords: Machine-to-Machine (M2M) Communication, Internet of Things (IoT), Forecasting, Machine Learning, Scheduling, Massive Access Problem

ÖZ

NESNELERİN İNTERNETİ İÇİN MAKİNE ÖĞRENMESİ TABANLI ÇOK ÖLÇEKLİ BÜTÜNLEŞİK TAHMİNLEME-ÇİZELGELEME

NAKIP, Mert
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği
Danışman: Doç. Dr. Volkan RODOPLU
Yardımcı Danışman: Prof. Dr. Cüneyt GÜZELİŞ
Temmuz 2020

Nesnelerin İnternetinin (IoT) Devasa Erişim Problemi, çok sayıda IoT cihazının kablosuz erişimini, kablolu altyapıya sağlama sorunudur. Bu tez çalışmasında, Orta Erişim Kontrol katmanındaki Devasa Erişim Problemini çözmek için atanmış bir IoT ağ geçidinde ortak tahmin-çizelgeleme için Çok Ölçekli Algoritma geliştirmekteyiz. IoT veri akışının rastgele olduğunu kabul eden Devasa Erişim Problemine yönelik mevcut yaklaşımların aksine, algoritmamız, Çok Katmanlı Algılayıcı mimarisini kullanarak IoT cihazlarının yaklaşan trafiğini tahmin eder ve bu tahminlere göre yer-uydu bağı (uplink) kablosuz kanalını önceden tahsis eder.

Öngörülü kaynak tahsisi algoritmalarının Devasa Erişim Problemini çözmek için pratikte uygulanabileceğini göstermek amacıyla, önce bağımsız bir IoT cihazının trafik oluşturma modelinin öngörülebilirliğini saptadık. Bunun için, Makineden Makineye iletişimdeki bağımsız IoT cihazlarının trafik üretim modellerini tahmin etme problemi üzerine Özyineli Tümlevli Kayan Ortalama, Çok Katmanlı Algılayıcı, 1 Boyutlu Evrişimsel Sinir Ağı ve Uzun-Kısa Süreli Bellek modellerinin karşılaştırma çalışmasını sunmaktayız. Tahmin modellerinin performansını karşılaştırmak için simetrik Ortalama Mutlak Yüzde Hatasını ölçmekteyiz. IoT trafiğini dört sınıfa ayırmaktayız: Sabit Bit Periyodik (FBP), Değişken Bit Periyodik (VBP), Sabit Bit Aperiyodik (FBA) ve Değişken Bit Aperiyodik (VBA). Simülasyon sonuçlarımızda, Uzun-Kısa Süreli Bellek modelinin VBP sınıfındaki cihazlar için diğer tüm modellerden önemli ölçüde daha başarılı olduğunu göstermekteyiz. Ek olarak, Uzun-Kısa Süreli Bellek modelinin Çok Katmanlı Algılayıcı ve 1 Boyutlu Evrişimsel Sinir Ağı ile FBA sınıfı için simetrik Ortalama Mutlak Yüzde Hatasında neredeyse aynı performansa sahip olduğunu göstermekteyiz. IoT cihazı başına eğitim süresi, Uzun-Kısa Süreli Bellek için en yüksek olmakla birlikte, tüm öngörme modellerinin pratik uygulama için makul eğitim süreleri vardır.

Tahminleme sonuçlarımız, IoT verilerinin oldukça öngörülebilir olduğunu gösterdiğinden, IoT'nin Devasa Erişim Problemini hafifletmek için Orta Erişim Kontrol katmanı için öngörülü bir kaynak ayırma algoritması olan Bütünleşik Tahminleme-Çizelgeleme sistemini önermekteyiz. Bütünleşik Tahminleme-Çizelgeleme sistemimiz, her bir IoT cihazının trafik üretim örüntüsünü tahmin eder ve bu cihazların iletimini önceden çizelgeler. Bütünleşik Tahminleme-Çizelgeleme sisteminin en yüksek ağ çıktısını elde ettiği yerel optimal tahminleme modelini seçmek için Özyineli Tümlevli Kayan Ortalama, Çok Katmanlı Algılayıcı ve Uzun-Kısa Süreli Bellek modellerini karşılaştırmaktayız. Bu tahminleme modelleri altındaki ağ veriminin karşılaştırılması, Bütünleşik Tahminleme-Çizelgeleme sistemi için tahminleme modelinin optimum seçiminin büyük ölçüde ağda bulunan farklı IoT cihazı sınıflarının oranlarına bağlı olduğunu ortaya koymaktadır. Ağ simülasyonlarımız, Bütünleşik Tahminleme-Çizelgeleme sisteminin tahminleme-çizelgeleme pencere boyutu 1800 saniye olarak ölçeklenirken, 1 saniyenin altındaki bir toplam yürütme süresi ile 1000 cihaza kadar destekleyebildiğini göstermektedir.

Bütünleşik Tahminleme-Çizelgeleme sistemini, daha uzun tahminleme-çizelgeleme pencere boyutuna sahip çok daha fazla sayıda cihazı destekleyecek şekilde ölçeklendirmek için, IoT uygulamalarının gecikme kısıtları ve IoT cihazlarının asgari trafik oluşturma aralıkları olarak belirlenen birden çok zaman ölçeğinde çalışan Çok Ölçekli Algoritmayı önermekteyiz. Algoritmamızın çok ölçekli yapısı, simülasyonlarımızda 6650'ye kadar IoT cihazını desteklemek için ölçeklenebilir zaman ve alan karmaşıklığı sağlamaktadır. Çok Ölçekli Algoritmanın verimliliği ve enerji tüketimi Rezervasyon Tabanlı Erişim Engelleme, Ortalama Yüke dayalı Öncelik ve Gelişmiş Öngörülü Sürüm Veri Bloğu Odaklı protokollerle karşılaştırılmaktadır ve Çok Ölçekli Algoritmanın 3000 üzerindeki cihaz sayısı için bu algoritmalardan önemli ölçüde daha iyi çalıştığını göstermekteyiz. Ayrıca, Çok Ölçekli Algoritmanın yüzde kontrol ek yükünün % 1.5'in altında kaldığını göstermekteyiz.

Bu tezde, sonuçlarımız IoT Ağ Geçitlerinde çok sayıda IoT cihazını işlemek için ölçeklenebilir bütünleşik tahminleme-çizelgeleme sistemleri oluşturmanın yolunu açmaktadır.

Anahtar Kelimeler: Makinelerarası (M2M) İletişim, Nesnelerin İnterneti (IoT), Tahminleme, Makine Öğrenmesi, Çizelgeleme, Devasa Erişim Problemi

ACKNOWLEDGEMENTS

During the progress of this work, I have been positively affected and motivated by many people, who have helped and supported this thesis in direct or indirect ways. First, I would like to thank my advisor Assoc. Prof. Volkan Rodoplu for his advice, direction and support throughout my Master's studies. Not only have I learned a wide of range of technical subjects from him, but also he has shown me the road to perfection in any work that one may undertake. Second, I would like to thank my co-advisor Prof. Cüneyt Güzeliş for his endless mentorship in regard to both my professional and personal life. I would also like to express my special thanks to Prof. Mustafa Seçmen for his great support during both my undergraduate and graduate studies. In addition, I would like to thank Prof. Deniz Türsel Eliiyi for her contributions in the earlier phases of my thesis.

I would like to thank my fiancée Gizem Güler for always being there whenever I needed her. I would also like to thank my teammates and colleagues Baran Can Gül, Oytun Uzun, Alperen Helva, Roozbeh Qorbanian, Kubilay Karakayalı, Onur Çopur and Ayşegül Kahraman for their friendship, help and for the tasteful technical discussions. I would like to thank Berna Karayılan for being very helpful with all of the administrative work throughout my Master's studies. In addition, all of the faculty members in the Department of Electrical and Electronics Engineering at Yaşar University have taught and helped me tremendously, so I would like to thank each of them.

This thesis was supported by the TÜBİTAK (Scientific and Technological Research Council of Turkey) 2211 National Graduate Scholarship Program (which is originally entitled as "TÜBİTAK 2211 Yurt İçi Lisansüstü Burs Programı") under 2210C High-Priority Technological Areas Graduate Scholarship. I would like to acknowledge the following institutions and programs that have provided employment or financial support during my Master's studies: Yaşar University, and the TÜBİTAK 1001 Master's Scholarship in Project # 118E277. Furthermore, I would like to thank the European Commission for providing support under Grant No. 846077.

Finally, I would like to express my gratitude towards and respect for my family for the sacrifices that they have made for my education and for their everlasting support.

Mert NAKIP
İzmir, 2020

TEXT OF OATH

I declare and honestly confirm that my study, titled “MACHINE LEARNING BASED MULTI-SCALE JOINT FORECASTING-SCHEDULING FOR THE INTERNET OF THINGS” and presented as a Master’s Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

MERT NAKIP

Signature:

[illegible]

July 22, 2020

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYMBOLS AND ABBREVIATIONS

List of Abbreviations

Abbreviation	Meaning
IoT	Internet of Things
M2M	Machine-to-Machine
ML	Machine Learning
H2H	Human-to-Human
UE	User-Enabled
H2M	Human-to-Machine
M2H	Machine-to-Human
M2M	Machine-to-Machine
QoS	Quality of Service
4G	Fourth Generation
PRACH	Physical Random Access Channel
OSI	Open System Interconnection
MAC	Medium Access Control
LLC	Logical Link Control
ACB	Access Class Barring
NOMA	Non-Orthogonal Multiple Access
SIC	Successive Interference Cancellation
RL	Reinforcement Learning
UL	Unsupervised Learning
SL	Supervised Learning
JFS	Joint Forecasting-Scheduling
FBP	Fixed-Bit Periodic
VBP	Variable-Bit Periodic
FBA	Fixed-Bit Aperiodic
VBA	Variable-Bit Aperiodic
ARIMA	Autoregressive Integrated Moving Average
MLP	Multi-Layer Perceptron

Continued on the next page

Continuation of the List of Abbreviations

Abbreviation	Meaning
1D CNN	1-Dimensional Convolutional Neural Network
LSTM	Long-Short Term Memory
sMAPE	symmetric Mean Absolute Percentage Error
MLP-SSA	MLP-Structure Selection Algorithm
PAL	Priority based on Average Load
MSA	Multi-Scale Algorithm
RAB	Reservation-based Access Barring
ACB	Access Class Barring
E-PRV-BO	Enhanced Predictive Version Burst Oriented
PRV-PO	Predictive Version Packet Oriented
LTE	Long Term Evolution
MA	Moving Average
RH	Relative Humidity
NMHC	Non-Methane Hydrocarbon
NO_2	Nitrogen Dioxide
TPU	Tensor Processing Unit
LM	Levenberg-Marquardt
MSE	Mean Square Error
Adam	Adaptive Moment Estimation
LDR	Light Dependent Resistor
MLE	Maximum Likelihood Estimation
STD	Standard Deviation
FDD	Frequency Division Duplexing
LBB	Load Balancing over Blocks
JSTW	Job Scheduling with Time Windows
5D	5-Dimensional
ACK	Acknowledgment
ANN	Artificial Neural Network

List of Symbols used in Chapter 1, 2 and 3

Symbol	**Meaning**
G	Gateway
$\mathscr{C}_G$	Coverage area of G
$\mathscr{N}$	Set of IoT devices in $\mathscr{C}_G$
N	Number of IoT devices in $\mathscr{C}_G$
$\mathscr{N}$	Set of N IoT devices in $\mathscr{C}_G$
Δ_j	Latency constraint of burst j
$x_i[m]$	Number of bits generated by device i in MAC-layer slot m
η	Uplink cross-layer throughput
T_{sch}	Duration of the scheduling window (in seconds)
s_f^i	Time index relative to the current time, that corresponds to the f^{th} feature to the traffic generation pattern of device i
F_i	Set of all of the indices of the features selected for device i
K_i	Number of forecasts of device i
$\hat{x}_i[m{+}k]$	The k^{th}-step ahead forecast for device i at current time slot m.
p	Autoregression coefficient of ARIMA
d	Integration coefficient of ARIMA
q	Moving Average coefficient of ARIMA
z_a^p	Lower bound of the search range for the local optimal value of p
z_b^p	Upper bound of the search range for the local optimal value of p
z_a^d	Lower bound of the search range for the local optimal value of d
z_b^d	Upper bound of the search range for the local optimal value of d
E	Total number of neural layers of the MLP
n_e	Total number of neurons at layer e of the MLP
n_0^i	Number of inputs of the MLP
p_{drop}	Dropout probability of the dropout layer of each of the 1D CNN and LSTM models
c_{CNN}	Number of filters at the convolution layer of the 1D CNN
z_a^{CNN}	Lower bound of the search range for the local optimal value of c_{CNN}
z_b^{CNN}	Upper bound of the search range for the local optimal value of c_{CNN}
h_{lstm}	Number of lstm units in the lstm layer of the LSTM model
$h_{\bar{e}}$	Number of neurons in the fully connected layer $\bar{e}$ of the LSTM
z_a^{lstm}	Lower bound of the search range for the local optimal value of h_{lstm}
z_b^{lstm}	Upper bound of the search range for the local optimal value of h_{lstm}

Continued on the next page

Continuation the List of Symbols used in Chapter 1, 2 and 3

Symbol	Meaning
$z_a^{\bar{e}}$	Lower bound of the search range for the local optimal value of $h_{\bar{e}}$
$z_b^{\bar{e}}$	Upper bound of the search range for the local optimal value of $h_{\bar{e}}$
η_o	Throughput obtained under perfect forecasts
Φ	Throughput difference between η_o and η
Φ_{min}	Minimum throughput difference in MLP-SSA-Φ
a_j	Number of bits of burst j
p_j	Number of slots required to send burst j completely
r_j	Slot in which burst j is generated
d_j	Slot by the end of which all of the bits of burst j have to be sent to G
R_j	Data rate (in bits per second) at which burst j can transmitted to G
τ_{MAC}	Duration of a MAC-layer slot
$\mathscr{J}$	Set of all bursts across all devices in $\mathscr{N}$
$\tilde{a}_j[m]$	Number of bits of burst j that have note yet been sent G at current time slot m
$\tilde{\Delta}_j[m]$	Length of time that remains at the current time m until the delay deadline d_j of burst j
$\mathscr{J}_{active}[m]$	Set of “active bursts” namely, those bursts which have been generated but not yet completely processed and whose delay deadlines have not yet expired at the current time m
$\gamma_j[m]$	Effective load of burst j at time m
$\mathbf{C}$	Capacity vector whose entry j is equal to $R_j\tau_{MAC}$
T_i	Traffic generation period of device i (in seconds)
T_i^{fc}	Execution time of forecaster of device i
T^{bank}	Execution time of the bank of parallel forecasters
T_{min}	Earliest generation time over all of the bursts in the upcoming scheduling window

List of Symbols used in Chapter 4 - Multi-Scale Algorithm

Symbol	Meaning
G	Gateway
$\mathscr{C}_G$	Coverage area of G

Continued on the next page

Continuation of the List of Symbols used in Chapter 4 - Multi-Scale Algorithm

Symbol	**Meaning**
$\mathscr{N}$	Set of IoT devices in $\mathscr{C}_G$
N	Number of IoT devices in $\mathscr{C}_G$
Δ_j	Latency constraint of burst j
$X_i[k]$	Number of bits generated by device i in MAC-layer slot k
η	Throughput
T_{sch}	Duration of the scheduling window (in seconds)
$\mathscr{A}_i$	Set of uplink MAC-layer slots allocated to device i
$\hat{X}_i[k]$	Forecast future traffic pattern of device i
T_i	Traffic generation period of device i (in seconds)
T_i^{af}	Duration of the accurate forecasting window of device i (in seconds)
V_l	Number of non-overlapping contiguous set of blocks at level l in Multi-Scale Forecasting
M_l	Number of blocks in each of the V_l sets at level l
K_i	Number of accurate forecasts of device i
T_{min}^{af}	Minimum accurate forecasting window duration of all devices
τ_{MAC}	Duration of a MAC-layer slot
S_B	Set of subblocks of block B
$\hat{X}_i(b)$	Forecast number of bits of device i in block b at level l
$\tilde{T}_{il}^{af}$	Length of the absolute time interval of accurate forecasts of device i's traffic at level l (in seconds)
B_l	Block duration at level l (in seconds)
$T_{sch}^{(l)}$	Duration of the scheduling window at level l (in seconds)
$\tilde{\Delta}_q$	Latency constraint of bin q after quantization
Q	Total number of bins in latency quantization
a_j	Number of bits of burst j
u_j	Binary decision variable that is 1 iff burst j is completed
y_{jk}	Binary decision variable that is 1 iff burst j appears in slot k
p_j	Number of slots required to send burst j completely
$\mathscr{K}^+$	Set of slots for which at least one y_{jk} has been defined
$\mathscr{J}$	Set of all bursts over the scheduling window
r_j	Slot in which burst j is generated
d_j	Slot by the end of which all of the bits of burst j have to be sent to G
C_i	Maximum number of bits that device i can send to G in a slot

Continued on the next page

Continuation of the List of Symbols used in Chapter 4 - Multi-Scale Algorithm

Symbol	**Meaning**
$H^{(l)}$	Maximum of the total number of bits in each block at level l
$\hat{X}_j^{(l)}$	Forecast of the number of bits of job j at level l
$r_j^{(l)}$	Block at level l in which job j is generated
$d_j^{(l)}$	Block at level l by the end of which all of the bits of job j must be successfully delivered
z_{jb}	Binary decision variable that is 1 iff job j is assigned to block b
$p_j^{(l)}$	Number of MAC-layer slots required to transmit all of the bits of job j that appears at level l
$\mathscr{J}^{(l)}$	Set of all jobs at level l over all devices in $\mathscr{N}$
$\mathscr{B}_+^{(l)}$	Set of blocks at level l for which at least one z_{jb} has been defined
$\mathbf{W}$	Set of fixed forecasting parameters (“weights")
$\mathbf{X}$	Set of past $\{X_i[k]\}$ for all i
$\mathbf{S}^{(\mathbf{B})}$	Schedule matrix for block **B**
R_b^l	Subblock b at level l
$T_i^{(max)}$	Maximum observed period of the empirical autocorrelation function of device i's traffic
E	Total number of neural layers of the MLP forecaster
p_{RAB}	Access probability of Reservation-based Access Barring
O_i	Number of slots reserved for device i
D_{max}	Maximum delay of the Predictive Version Packet Oriented (PRV-PO) protocol
Δt_{max}	Maximum jitter of the PRV-PO protocol
Δt_{max}^j	Maximum jitter of burst j of Enhanced Predictive Version Burst Oriented (E-PRV-BO) protocol
$\mathscr{E}$	Total transmit energy consumption of all devices divided by the total number of bits in successfully delivered bursts

Chapter 1
INTRODUCTION

The main goal of this thesis is the design of a machine-learning-based multi-scale joint forecasting-scheduling system to solve the Massive Access Problem of the Internet of Things (IoT). In this chapter, our goal is to clearly state the problem that will be addressed in this thesis and to review the background that will be required for the upcoming chapters. First, we review M2M communication and IoT in order to demonstrate the increasing trend of the number as well as the growing range of IoT devices. Second, we describe the Massive Access Problem of M2M communication. Third, we describe how Machine Learning (ML) can be used to solve the Massive Access Problem. In addition, we review the ML algorithms that we shall employ in order to solve the Massive Access Problem. Fourth, we state the assumptions that underlie the rest of this thesis. Fifth, we describe the relationship between our study and the state of the art. Finally, we summarize how this thesis advances the state of the art and give an outline for the rest of this thesis.

1.1 M2M Communication and IoT

The Internet has been growing rapidly in the last five years. According to Cisco's estimates (Cisco, Mar. 2020), approximately 30 billion devices are expected to be on the Internet by the year 2023. Most of these devices are expected to be machines that operate with minimum or no human intervention (Ghavimi & Chen, 2015).

In the current networking systems, there are four main types of communication. The first type of communication is Human-to-Human (H2H) communication, which enables a wired or wireless connection between two or more User-Enabled (UE) devices. Some prominent examples of H2H communication are the mobile phone calls, video conferences, and live video streams. The second type of communication is Human-to-Machine (H2M) communication, where the UE device (which is operated by a human) communicates with at least one device (which is not operated by a human user). In the majority of applications, this device is a web server. The third type of communication is Machine-to-Human (M2H) communication. In M2H communication, a device transmits data to UE devices without or with minimum human interaction. The communication of all of the automatic notifications to smart phones or computers are in M2H communication. The fourth type of communication is Machine-to-Machine (M2M) communication, which constitutes the scope of this thesis. The communication between two or more devices that operate without any human intervention is referred to as M2M communication.

The M2M communication is a promising technology for the smart city concept of the near future (Hasan, Hossain, & Niyato, 2013). An M2M communication system has diverse applications within a large-scale network and a massive number of interconnected devices (Lo, Law, & Jacobsson, 2013). The most common examples of M2M communication are as follows (Lawton, 2004): smart utility meters, smart garbage bins, surveillance and security, infrastructure management, smart parking system, traffic monitoring, and electronic health by wearable devices (Glaros & Fotiadis, 2005). In M2M communication, devices are “smart (intelligent) devices”. In this type of communication, a large number of smart devices can communicate each other and make collaborative decisions without human intervention (Chen, Wan, & Li, 2012) (Igarashi, Ueno, & Fujisaki, 2012) to improve the efficiency of labor time (Verma et al., 2016).

Most of the M2M technologies support a network of a massive number of devices, which are collectively known as “Internet of Things” (IoT) (Glitho, 2011). In recent years, IoT has wide-range application areas, which aim to increase the quality of human life (Bello & Zeadally, 2019)(Fortino, Russo, Savaglio, Viroli, & Zhou, 2017). Some examples of these areas are cities, homes, universities, industrial factories, organizations, agriculture environments, hospitals, security and safety systems, and health-care centers (Muralidharan, Roy, & Saxena, 2018)(Terroso-Saenz, González-Vidal, Ramallo-González, & Skarmeta, 2019)(Gubbi, Buyya, Marusic, & Palaniswami, 2013)(Miorandi, Sicari, De Pellegrini, & Chlamtac, 2012). Since these applications are the basis of the smart cities of the near future (Zanella, Bui, Castellani, Vangelista, & Zorzi, 2014), the number of applications are increasing every day.

From a networking perspective, the main challenge that IoT brings is the satisfaction of the diverse set of Quality of Service (QoS) constraints of IoT applications (Asghari, Rahmani, & Javadi, 2019). QoS constraints typically drive the determination of priorities during network resource allocation for each specific type of data on the network. Furthermore, QoS aims to address a heterogeneous set of performance objectives such as maximizing network throughput, and reducing the energy consumption, packet loss, latency and jitter. One of the most important QoS constraints of IoT devices is the delay constraint for a delay-sensitive IoT application: For each such application, there is a specific delay constraint by which the IoT must successfully transmit its data to the destination. Some examples of such delay constraints are as follows: 5 ms for autonomous vehicles; 10 ms for teleprotection in a smart grid network; seconds to minutes for smart home applications; and approximately 1 hour for smart utility meters (Osseiran, Monserrat, & Marsch, 2016)(Ict, 2013).

According to Ericsson (Zaidi, Hussain, Hogan, & Kuhlins, Jan. 2019), IoT devices can be segmented into four categories: Broadband IoT, Critical IoT, Industrial Automation IoT, and Massive IoT. In Broadband IoT, the IoT devices use the enhanced mobile broadband for IoT to provide large volumes of data transfer with high data rates and low latencies. Examples of Broadband IoT are the applications of drones, Augmented Reality, and Virtual Reality. In Critical IoT, the communication is time-critical. The most common examples for the Critical IoT are the electronic health applications and autonomous cars. In Industrial Automation IoT, in order to enable the real-time automation, the seamless integration of cellular connectivity into the wired industrial infrastructure is targeted. In Massive IoT, in addition to meeting a diverse range of QoS requirements, including delay or energy consumption constraints, the successful connection from the massive number of IoT devices to a cellular base station or a gateway constitutes the main goal of the applications in this category. The last category, Massive IoT, constitutes the main focus of this thesis.

1.1.1 Massive IoT

In the Massive IoT segment, the devices have much simpler and much more low-cost hardware than the IoT devices in other segments. Thus, the data rates are lower than the devices in the remaining segments as well. In addition, in Massive IoT, the data volumes are lower than those in other segments; however, the number of devices that fall into the coverage area of a base station is much higher than that in the Broadband, Critical and Industrial Automation IoT segments.

In Table 1.1, we show sample applications for Massive IoT devices. Note that the majority of the examples in this table are adapted from (Kuhlins, Rathonyi, Zaidi, & Hogan, Jan. 2020). Ericsson estimates that 52% of all of the IoT devices will fall into Massive IoT segment at the end of 2025 (Ericsson, Nov. 2019).

Table 1.1. Sample applications for massive IoT

Application Area	**Application Name**
Transportation and Logistic	Fleet Managment
Agriculture	Climate Monitoring
	Service Monitoring
Environment	Flood Alerting
	Pollution Monitoring
Industrial	Process Control
	Maintenance Monitoring
Utilities	Smart Metering
	Smart Grid Management
Smart Cities	Parking Sensors
	Smart Bins
	Smart Lighting
	Traffic Controling
Smart Buildings	Smoke Detectors
	Security Systems
	Home Automation

1.2 Massive Access Problem

The number of IoT devices in the Massive IoT segment is predicted to increase almost exponentially at least up to 2025. The "Massive Access Problem" is the problem of enabling the wireless access of a massive number of IoT devices to the wired infrastructure. Among diverse IoT device types that include fleet management devices as well as outdoor data sensors for smart cities, smart meters alone are estimated to reach a density of 5,000 to 35,000 per cell in urban areas (Vodafone, Apr. 2010). In Fig. 1.1, a representative visualization of the increasing massive number of IoT devices is given. In this figure, the dashed red lines show the IoT devices that are attempting to connect to the base station via wireless links.

The Massive Access Problem has been singled out as a key challenge that requires a solution in order to enable the IoT revolution. Since current cellular systems (the fourth generation (4G) wireless systems) would be unable to handle such a massive number of devices due to the Physical Random Access Channel (PRACH) overload (Ghavimi & Chen, 2015)(Zanella et al., 2013), the solution of the Massive Access

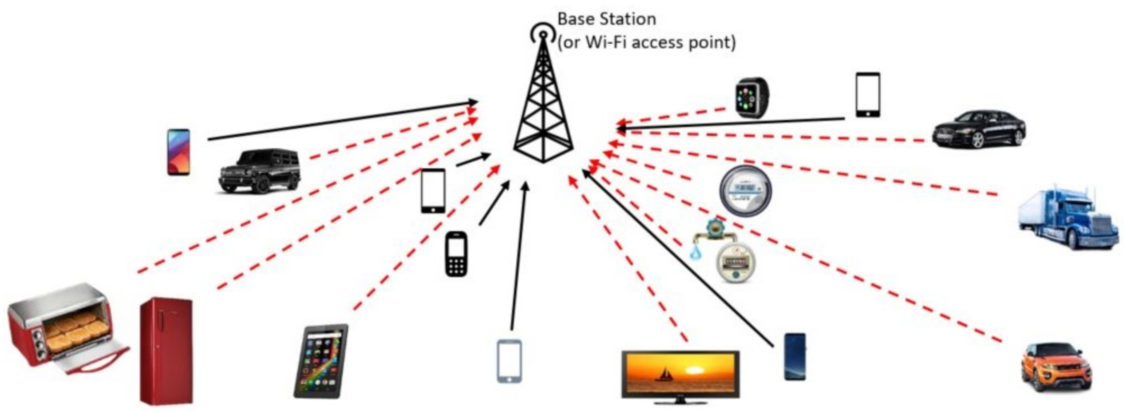

Figure 1.1. An illustration of the massive number of IoT devices that attempt to access a single base station or access point

Problem requires novel approaches that enable wireless connectivity in order to satisfy the QoS requirements of the IoT devices. The scalable solutions developed in this thesis to solve the Massive Access Problem constitute an important step in helping launch the IoT revolution.

In the rest of this section, we shall review the key terms and models that will aid in describing the Massive Access Problem. First, we review the Open Systems Interconnection (OSI) model, which is the standard communication protocol stack for modern networks. Second, we describe the Medium Access Control sublayer of the Data Link Control layer in this protocol stack. Third, we describe the Physical Random Access Channel in Long-Term Evolution cellular systems. Fourth, we describe the problem of collisions that occurs when a massive number of IoT devices seek to access this channel. Based on these concepts, we describe the Massive Access Problem. Finally, we outline the existing approaches taken to solve the Massive Access Problem.

1.2.1 Open Systems Interconnection Model

The current communication systems are characterized and standardized by the OSI model. As shown in Fig. 1.2, the OSI model consists of 7 layers, which are the (1) physical layer, (2) data link layer, (3) network layer, (4) transport layer, (5) session layer, (6) presentation layer, and (7) application layer.

The physical layer enables the transmission or reception of the bits rather than the data packets and translation of the logical communication requests into hardware-specific operations (Shaw, 2017).

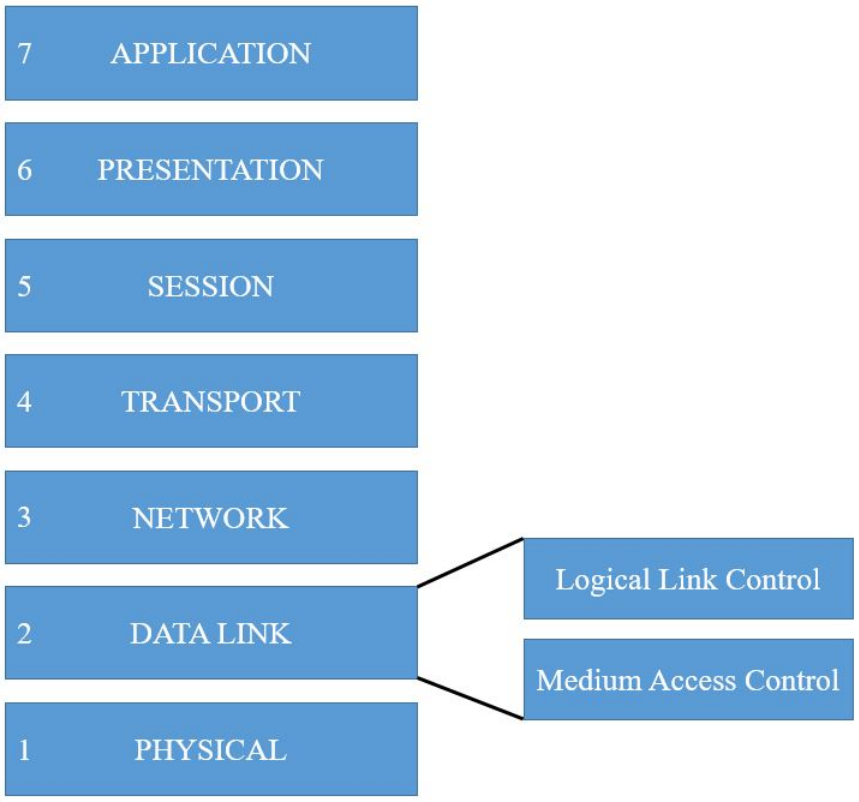

Figure 1.2. Open Systems Interconnection (OSI) model

At the data link layer, data transmission between two connected nodes is provided. The response to the service requests and the allocation of resources are handled at this layer. In addition, the errors that occur at the physical layer might be detected and corrected. The data link layer consists of two sublayers which are the Medium Access Control (MAC) sublayer (which is called the MAC layer, for short) and the Logical Link Control (LLC) sublayer (which is called the LLC layer, for short). The MAC layer determines which device is allowed to access to the medium at any time.[1] The LLC layer multiplexes protocols, and provides the data flow control, acknowledgment and error notification. Since, we design the resource allocation algorithms for the MAC layer, we will give the further details on the resource allocation at MAC layer after we briefly review the rest of the layers of the OSI model.

At the network layer, which is the third layer in Fig. 1.2, the transmission of the network packets from a source device to a destination device over the networks is

[1]Typically, the medium refers to the frequency channels that are available for communication. It may also include the spatial channels, as in Multiple Input Multiple Output (MIMO) systems, which are not addressed in this thesis.

provided (Zimmermann, 1980). In the OSI model, this layer responds to the connection requests from the transport layer and issues them to the data link layer. The transport layer provides the end-to-end connections, reliability, and data flow control (Braden, 1989). The session layer opens, closes and manages the sessions between applications. This layer is responsible for checkpointing as well as recovery in each session, and combines or synchronizes the information of several streams. At the presentation layer, the formation, representation, and the encryption of data are carried out in order to fix the syntactical differences in data for the further operations at the application layer (Grigonis, 2000). At the application layer, the communication protocols and the interfaces that are used by the host, are specified. At this layer, the connection between the host application and the layers of the OSI model is provided.

1.2.2 MAC Layer

We now explain the MAC layer briefly, since the analysis and the design of the proposed algorithms in this thesis are performed for this layer. The MAC layer is the sublayer of the data link layer, which is the second layer of the OSI model. The MAC layer is responsible for the mapping between the logical and the transport channels, error correction, physical addressing, and allocation of the air interface resources for both uplink and downlink communication (Alpern, 2009). By the allocation of the resources, the MAC layer controls the access to the transmission medium. It is also responsible for the initiation of data retransmission when a collision has occurred.

The MAC layer uses the transport channels that are exported by the physical layer. The transport channels of the physical layer are as follows: Physical Down Link Shared Channel, Physical Broadcast Channel, Physical Multicast Channel, Physical Uplink Shared Channel and PRACH (Samaoui, El Bouabidi, Obaidat, Zarai, & Mansouri, 2015).

PRACH:

The PRACH is a periodic sequence of uplink time-frequency resource blocks, which are used by devices for random access in LTE networks. The random access refers the request of connection setup to the base station (Zaidi et al., 2018). Note that in current LTE networks, the period and the number of resource blocks of the PRACH is defined by the PRACH configuration index (Bouzouita, Hadjadj-Aoul, Zangar, Tabbane, & Viho, 2015).

The PRACH provides the synchronization between the base station and the devices, and it reserves the resources for data. In order to access the PRACH, a device transmits a preamble, which is a specific pattern selected randomly among a *predefined* set of preambles (Condoluci, Dohler, & Araniti, 2017). Note that each preamble in this set is orthogonal to other preambles. Since the set of preambles is a predefined set with constant number of preambles, it is possible that more than one device can select the same preamble. If two or more devices transmit the same preamble on the same time-frequency resource block of the PRACH, a collision occurs. In this case, none of the devices can access the PRACH. The collision problem on the PRACH is detailed in the next subsection.

1.2.3 Collision Problem on PRACH

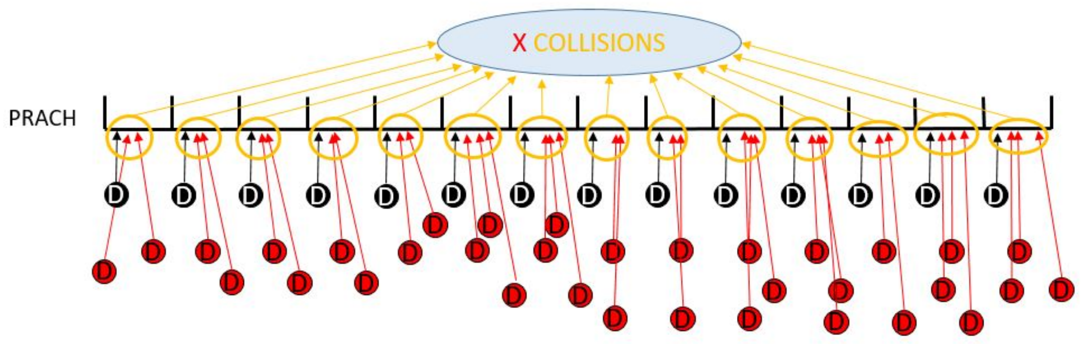

Figure 1.3. Visualization of collisions on the PRACH in the case in which a massive number of IoT devices attempt to gain access to a base station or access point

Even if the number of channels, the duration of each MAC-layer slot and the uplink data rate of devices are sufficient to handle the massive number of IoT devices the uplink collision may still occur at the PRACH, as shown in Fig. 1.3. In this figure, the symbolic notation D inside a circle denotes an IoT device that attempts to access the uplink time-frequency resource block of the PRACH, and this access attempt is shown by an arrow. In addition, the red circles represent the devices that are being added as time proceeds. That is, the number of devices increases with time. Since the number of IoT devices increases rapidly, the number of red circles in Fig. 1.3 and the probability of transmission of the same preamble to the same block of the PRACH by more than one device also increases.

As we shall see, one of the key aspects of all of the scheduling schemes that we propose in this thesis is collision-free scheduling, which refers to scheduling the uplink transmissions of devices in a manner that is completely free of any collisions. In contrast with our approach, almost all of the current approaches to the Massive Access Problem involve access in the presence of collisions. (The few exceptions will be noted in the next subsection.) Since collisions decrease channel utilization and result in excessive transmit energy expenditure, collision-free scheduling is highly preferable if it can be achieved.

1.2.4 Current Approaches to Solve the Massive Access Problem

The current approaches to the solution of the Massive Access Problem can be classified into six categories: (1) Access Class Barring (ACB) schemes (Ghavimi & Chen, 2015)(Lien, Liau, Kao, & Chen, 2012)(Lin, Lee, Cheng, & Chen, 2014)(Jin, Toor, Jung, & Seo, 2017)(J. Liu, Song, et al., 2017)(Tello-Oquendo, Leyva-Mayorga, et al., 2018)(Tello-Oquendo, Pacheco-Paramo, Pla, & Martinez-Bauset, 2018), (2) Cognitive M2M communication schemes (Aijaz & Aghvami, 2015)(Aijaz, Ping, Akhavan, & Aghvami, 2014), (3) Game-theoretic approaches (Pang, Chao, Lin, & Wei, 2014), (4) Clustering-based (Park, Kim, & Har, 2015)(Liang, Xu, Cao, & Jia, 2018) and rate-adaptive approaches (Si, Yang, Chen, & Xi, 2015), (5) Spread-spectrum, Non-Orthogonal Multiple Access (NOMA) (Shirvanimoghaddam, Dohler, & Johnson, 2017) and Successive Interference Cancellation (SIC) approaches (Zanella et al., 2013), and (6) CSMA/CA or slotted-ALOHA-based (Y. Liu, Yuen, Cao, Hassan, & Chen, 2014)(Shahin, Ali, & Kim, 2018) and collision-aware (Alavikia & Ghasemi, 2018) protocols. In all of these approaches, IoT traffic at the MAC layer is modeled by “random arrivals”. Our key observation is that the performance of all of these approaches is limited by this assumption at the MAC layer.

The assumption of “random arrivals” has so far dictated that only *reactive* MAC-layer solutions to the Massive Access Problem are possible. In contrast with these solutions, in this thesis, we show that when the traffic patterns of IoT devices are predicted via machine learning, *proactive* solutions to the Massive Access Problem become viable. In the next section, we shall focus on machine learning as a novel enabler of a solution to the Massive Access Problem.

1.3 Machine Learning for the Solution of Massive Access Problem

ML refers to the learning carried out by an algorithm, which learns the tasks that are required and improves its own performance on these tasks via accumulation of experience. The experience in ML is generally provided by the collected experimental dataset. By the design of the structure of an ML algorithm, a mathematical model is created. Thus, the ML algorithm computes the parameters of this mathematical model based on the dataset, which is called the "training data". By using computed parameters, the ML algorithm aims to make predictions or decisions, where the algorithm is not explicitly programmed to carry out these operations (Bishop, 2006).

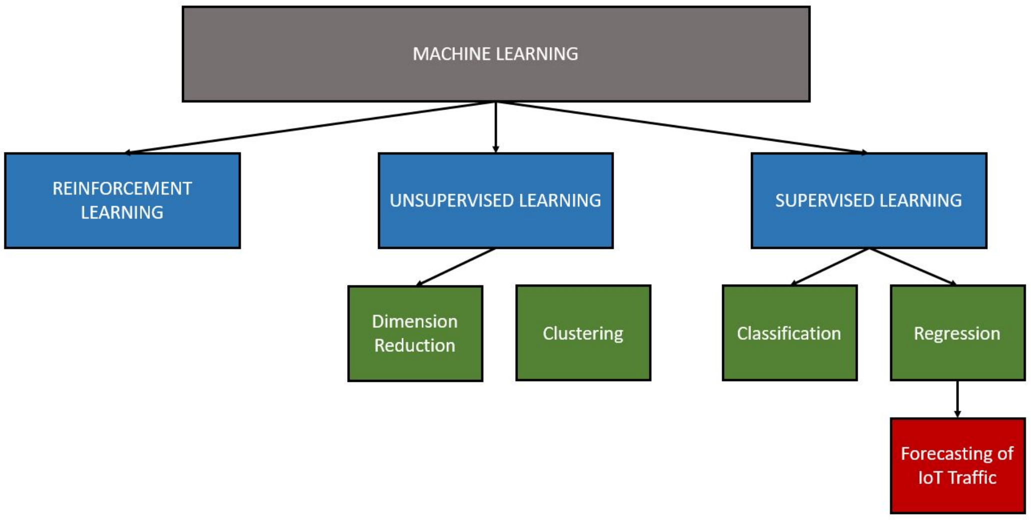

Figure 1.4. Classification of Machine Learning problems

In Fig. 1.4, we present the broad classification of ML problems. The diagram also identifies the problem class of the IoT data forecasting. As shown in this figure, in order to review the ML algorithms that are used commonly, we can split the learning algorithms of the ML models into three classes as follows: (1) reinforcement learning (RL), (2) unsupervised learning (UL), and (3) supervised learning (SL).

The ML algorithms that fall in the RL class aim to maximize the cumulative reward or minimize the cumulative penalty of actions that are taken by software agents. In this class, there is no requirement for the training data with the desired output samples. In addition, in this class, the algorithms are used when the exact parameters of the model are not known, or the exact models are infeasible (Wiering & Van Otterlo, 2012). The most common examples of RL algorithms are the real-time decision making systems and the actively learning robotic systems.

The algorithms in the UL class require the dataset for the learning stage (which is carried out on the training data), which is operates only on the input sample; that is the algorithms in this class do not require the set of labeled samples (Hinton, Sejnowski, Poggio, et al., 1999). The most common applications of the algorithms in the SL class might be split into the dimension reduction of the data and clustering problems. The feature elimination algorithms are examples of unsupervised dimension reduction. In addition, a recommendation system is an example of the unsupervised clustering.

In the SL class, the training data is provided as input-output pairs; that is, each sample in the dataset is fully and clearly labeled (Russel, Norvig, et al., 2013). In this class, an algorithm optimizes its parameters in order to maximize the performance on the training set. The problems that are solved by using SL fall into three classes, which are the active learning, classification and regression (Alpaydin, 2020). One of the reasons for using active learning is that when the subset of the dataset is unlabeled, manual labeling is an expensive operation. In this case, the algorithm actively uses the information source (which is called a teacher) in order to label each of the unlabeled samples. The classification problem is a one of categorizating data into predefined categories. One of the most popular examples of the classification problems is the image recognition problem. Furthermore, the regression problem is to find the most accurate fit that generalizes the function of the labeled training set. The models that solve the regression problems are generally used for the prediction and the forecasting. Forecasting is a sub-discipline of prediction in which the problem is to make a prediction regarding the future based on time series data. That is, in a forecasting problem, at least one input of the model is the past sample of the time series and the output of the model is another sample. In this thesis, since we aim to compute the future samples of the traffic generation pattern of individual IoT device, our problem falls into the forecasting problem class.

1.3.1 Forecasting of the Traffic Generation Pattern of an IoT Device

One of the major goals of this thesis is to show that the traffic generation patterns of M2M communication are far more predictable than has been assumed so far in the past work on the Massive Access Problem. While the random arrival model may be well-suited to H2H or H2M traffic, it is not necessarily suitable for M2M traffic since both the amount and the timing of machine-generated traffic can be highly predictable.

Predictability of M2M traffic has far-reaching implications for the design of practical access schemes. For example, if all IoT traffic could be predicted accurately, then a base station or IoT Gateway could utilize a joint forecasting-scheduling architecture in which it predicts the future traffic of all IoT devices over a scheduling window and would schedule all of the traffic collision-free in advance. Thus, for the solution of the massive access problem, the accurate knowledge on the traffic generation pattern of the IoT devices has a key role.

In this thesis, in order to be able to achieve an efficient allocation of systems resources across all IoT devices in a given coverage area, an IoT gateway forecasts the future traffic generation pattern of each device based on its past traffic generation pattern. The traffic generation pattern of an IoT device is a time series dataset that is comprised of the collection of the number of bits that are generated by that device over time. In addition, all of the features of the dataset potentially vary with IoT devices. The examples of these features are the range of the number of bits, the seasonal trends, and the number of samples. Furthermore, in order to use the ML models to forecast the traffic generation pattern, we convert the time series data to input-output pairs. We apply this conversion in order to be able to use supervised learning.

This completes our overview of forecasting for M2M traffic. In the next section, we shall detail the general assumptions that underlie the design of wireless communication systems for IoT in our treatment in this thesis.

1.4 Assumptions on the Wireless Communication System of IoT

In the mathematical framework of this entire thesis, we assume that there is a single IoT Gateway, denoted by G, and a set of N IoT devices, denoted by $\mathscr{N}$, each of which has a direct wireless link to G. (The IoT Gateway may be co-located with or have a fast wireless link to a cellular base station or Wi-Fi access point.) Furthermore, each IoT device ("device" for short) in the coverage area $\mathscr{C}_G$ of G is assumed to be associated with G at all times. (We do not model loss or re-gain of association.) We assume that each device in $\mathscr{N}$ remains in $\mathscr{C}_G$ at all times. This models the case where each device in $\mathscr{N}$ is static (i.e. non-moving) or is mobile but remains in $\mathscr{C}_G$.[2] In our future work, we shall address the cases in which mobile devices change coverage areas and devices are added to or removed from the network (Section 4.5).

[2]The latter situation corresponds e.g. to those smart factories in which each mobile device remains in the coverage area of a particular IoT gateway.

We define a "burst" as a set of logically grouped data in close proximity in time, generated by a single device. For example, the information bits all of which correspond to a single reading from a smart meter, and those that correspond to a single GPS coordinate reading from a fleet management device are examples of bursts. The aim of IoT communication is to deliver the entire burst; that is, the delivery of only a part of a burst would be useless. We assume a cross-layer design in which each burst is identified as such to the MAC layer. Furthermore, as part of the overall end-to-end delay constraint budget to the M2M server, each burst j to G has a MAC-layer latency constraint, denoted by Δ_j, which depends on the application that generates the burst.

We denote each MAC-layer slot (or simply "slot") at G by k. We assume that uplink transmissions of all devices to G have been slot-synchronized. We define the "traffic generation pattern" of device i as the number of bits generated by i during MAC-layer slot k and denote it by the collection $\{X_i[k]\}$ over the slot index k. We assume that each IoT device generates at most one burst in each MAC-layer slot. We allow each burst to be sent "with preemption", which means that the transmission of the burst is allowed to be split over a non-contiguous set of time slots.

We assume that there is a single uplink channel at G that is to be allocated among $\mathscr{N}$. (We shall address the multi-channel case in our future work.) In order to schedule uplink transmissions without collision, each MAC-layer slot k at G can be allocated to at most a single device at a time.

We say that burst j has been "successfully delivered" if all of its information bits have been sent to G without any collision while satisfying the Δ_j latency constraint. We define the "uplink cross-layer throughput" η from $\mathscr{N}$ to G as the total number of bits in successfully delivered bursts divided by the total number of bits of traffic offered by $\mathscr{N}$ over a scheduling window whose duration is denoted by T_{sch}.

1.5 Relationship to the State of the Art

There is a marked difference between this work and the current approaches to the Massive Access Problem, which were categorized in Section 1.2.4. All of the papers in these six categories model the traffic generation process by "random arrivals". In contrast, the main focus of this work is that high network performance can be achieved by using forecasts of IoT device traffic.

In the recent literature, there have been a plethora of articles (Boutaba et al., 2018) (Chinchali et al., 2018) on network traffic forecasting. The key difference is that these articles forecast large-scale, aggregate traffic patterns (such as traffic volume, link load, path availability) whereas our work focuses on forecasting the future traffic of *individual devices* at the MAC layer. For the Massive Access Problem, it is essential that the forecast at the IoT Gateway be obtained *for each device* in order to be able to schedule the traffic of individual devices that share the same wireless resources. This is in sharp contrast with the assumptions of all the papers in this category. Furthermore, these past works have not developed a joint forecasting-scheduling methodology, whereas this constitutes a major focus of our work.

There have been only a few works (Petkov & Obraczka, 2012)(Petkov & Obraczka, 2011)(Edalat, Ahn, & Obraczka, 2016) in the literature that have taken a proactive approach to Medium Access Control. The key differences between these works and ours are as follows: (1) These works focus on H2H applications (such as Skype), whereas our entire focus in this work is on the M2M traffic of IoT. The traffic generation patterns of these two application domains are different. (2) For the subset of these works (Petkov & Obraczka, 2012)(Petkov & Obraczka, 2011) that also specify scheduling techniques, further differences are as follows: (a) They use a probabilistic channel scheduling technique, whereas we employ fast, deterministic scheduling techniques. (b) We solve the scheduling problem for IoT devices, which operate at widely different time scales, using a novel multi-scale joint forecasting-scheduling technique that achieves high network performance.

1.5.1 Contributions of this Thesis

The main contribution of this thesis is the development of a novel algorithm that operates at multiple time scales to perform joint forecasting and scheduling of IoT devices at the MAC layer. The second contribution of this thesis is the design of a Joint Forecasting-Scheduling (JFS) system, to be implemented at an IoT Gateway, that forecasts the future traffic generation pattern of each IoT device and schedules the traffic of these devices in advance without any collisions over a scheduling window. To the best of the authors' knowledge, the development of a joint forecasting-scheduling methodology at the MAC layer is completely novel. The third contribution of this thesis is that we show that the traffic generation patterns of M2M communication are far more predictable than has been assumed so far in the past work on the Massive Access Problem. While the random arrival model may be well-suited to H2H and H2M traffic, it is not necessarily suitable for M2M traffic since both the amount and the timing of machine-generated traffic can be highly predictable. Predictability of M2M traffic has far-reaching implications for the design of practical access schemes. The fourth contribution of this thesis is a comparison of the performance of JFS with respect to the forecasting scheme employed. Furthermore, in this thesis, we divided the IoT devices into four classes based on their traffic generation characteristics: (1) Fixed-Bit Periodic (FBP), (2) Variable-Bit Periodic (VBP), (3) Fixed-Bit Aperiodic (FBA), and (4) Variable-Bit Aperiodic (VBA). If an IoT device generates a constant number of bits over time in each generation instance, we categorize this device as “Fixed-Bit”; otherwise, we say that the device is “Variable-Bit”. Furthermore, if the device generates traffic at regularly spaced time instances, then we call the device “Periodic”; otherwise, we say that the device is “Aperiodic”.

1.6 Outline of this Thesis

The remainder of this thesis is organized as follows: In Chapter 2, we undertake a comparative study of four fundamental forecasting models for the forecasting of the traffic generation pattern of individual IoT devices in M2M communication. These four forecasting models are the (1) Autoregressive Integrated Moving Average (ARIMA), (2) Multi-Layer Perceptron (MLP), (3) 1-Dimensional Convolutional Neural Network (1D CNN), and (4) Long-Short Term Memory (LSTM). Furthermore, by fine-tuning the architecture and the parameters of each of these four models, we measure the best performance attainable by each model in the symmetric Mean Absolute Percentage Error (sMAPE) metric. In order to find the local optimal architecture of the MLP neural network, we propose the MLP−Structure Selection Algorithm (MLP-SSA). In this chapter, we analyze the trade-off between the sMAPE forecasting performance and the training time of the forecasting models.

In Chapter 3, first, we propose the JFS for the Massive Access problem of IoT. JFS system, which will be implemented in at an IoT Gateway, forecasts the future traffic generation pattern of each IoT device and schedules the traffic of these devices in advance without any collisions over a scheduling window. Such a system has the advantage of potentially achieving a high throughput and minimizing the average transmit energy spent by each IoT device, thereby prolonging the lifetime of battery-limited IoT devices. Second, in this chapter, we perform a comparison of the throughput performance of JFS system under three different forecasting schemes, which are the ARIMA, MLP, and LSTM models. Since we measure the network performance via the uplink cross-layer throughput in a full JFS system in our comparison, it provides a much more accurate measure of performance than traditional performance metrics for forecasting (such as Mean Square Error and symmetric Mean Absolute Percentage

Error) that do not directly quantify the impact of forecasting on scheduling. Furthermore, we propose a reservation-based heuristic scheduling algorithm, namely Priority based on Average Load (PAL), to allocate the MAC-layer slots for IoT devices. In addition, via our simulations, we estimate the computation time that would be required to implement JFS in an actual system that operates at an IoT Gateway.

In Chapter 4, first, we present the development of a novel Multi-Scale Algorithm (MSA) that operates at multiple time scales to perform joint forecasting and scheduling of IoT devices at the MAC layer. The MSA improves the throughput performance achieved by the JFS system, enables JFS to work with much longer scheduling windows and decreases the effects of the forecasting errors on the network performance. The main advantage of this approach is three-fold: (1) The amount of control signaling overhead can be kept to a minimum. (2) The delays that occur in traditional protocols due to contention, collisions and handshaking can be avoided. (3) If the forecasts are accurate, high network performance can be achieved via reservation-based scheduling.

Second, in Chapter 4, we propose Multi-Scale Forecasting with a specific implementation via the MLP model. We show that a MLP-based architecture for forecasting, combined with exact scheduling solutions, achieves high network throughput while satisfying the diverse delay deadlines of the IoT devices. Furthermore, we compare the throughput and energy consumption of MSA with those of three protocols: (1) Reservation-based Access Barring (RAB), which is a reactive, joint access-scheduling protocol that we have designed that combines features of Access Class Barring (ACB)(Ghavimi & Chen, 2015) and reservation-based scheduling(Yu, Navaratnam, & Moessner, 2013). (2) PAL, which is a proactive protocol that we have designed that utilizes non-preemptive priority scheduling(Stallings, 2009) such that the priority across devices is determined based on the time-averaged forecast remaining traffic load until the delay deadline. (3) Enhanced Predictive Version Burst Oriented (E-PRV-BO) protocol, which is a proactive protocol that enhances the Predictive Version Packet Oriented (PRV-PO) protocol in (Hammad, Moubayed, Primak, & Shami, 2017). Finally, we present a comparison of the performance of MSA against those of the above three protocols under the IEEE 802.11ah standard.

In Chapter 5, we summarize the contributions of this thesis. In addition, we discuss the work that we plan to undertake in the future in order to extend this thesis in new directions.

Chapter 2

FORECASTING OF THE IoT DEVICE TRAFFIC AT THE MEDIUM ACCESS CONTROL LAYER

2.1 Introduction

As discussed in Chapter 1, the Massive Access Problem of Machine-to-Machine communication refers to the problem of providing the uplink wireless access of a massive number of IoT devices to the wired infrastructure (Ghavimi & Chen, 2015). The majority of the network architectures under consideration for the wireless access of IoT devices are expected to provide direct connectivity on the uplink from each IoT device to a base station or IoT Gateway (Chen et al., 2012). However, among the many types of IoT devices such as fleet management devices and outdoor data sensors for smart cities, smart utility meters alone are estimated to reach a density of 5,000 to 35,000 per cell in urban areas (Vodafone, Apr. 2010). Providing direct uplink connectivity to such a high density of IoT devices constitutes a major challenge that must be overcome via the design of wireless access protocols.

The past approaches (Liang et al., 2018)(Lien et al., 2012)(Lin et al., 2014)(Jin et al., 2017)(J. Liu et al., 2017) to the Massive Access Problem have modeled the MAC-layer traffic of individual IoT devices via random arrivals. Under the random arrival model, the future traffic generation pattern of an IoT device cannot be predicted accurately based on the past traffic of that device. The random arrival model thus leads to the design of *reactive* MAC-layer protocols in which each IoT device contends for the channel whenever it has newly generated data.

Prior work (Ghavimi & Chen, 2015) has shown that 4G systems cannot handle the access attempts of IoT devices on the PRACH. As a result, a multitude of articles, such as (Tello-Oquendo, Leyva-Mayorga, et al., 2018) and (Tello-Oquendo, Pacheco-Paramo, et al., 2018), have proposed methods by which the PRACH overload problem in Long Term Evolution (LTE) systems can be solved. However, these past approaches have continued to assume random arrivals in modeling the traffic generation patterns of individual IoT devices. As a result, they have not been able to benefit from any prediction of the future traffic generation patterns of individual IoT devices.

The main thesis of this chapter[1] is that the traffic generation patterns of M2M communication are far more predictable than has been assumed so far in the past work on the Massive Access Problem. While the random arrival model may be well-suited to H2M or H2H traffic, it is not necessarily suitable for M2M traffic since both the amount and the timing of machine-generated traffic can be highly predictable. For example, many smart utility meters currently generate traffic in fixed amounts at regular intervals. As a result, their traffic can be predicted with 100% accuracy. The main purpose of this chapter is to explore whether predictability extends to other device classes and quantify the performance of the prediction schemes employed.

Predictability of M2M traffic has far-reaching implications for the design of practical access schemes. For example, if all IoT traffic could be predicted accurately, then a base station or IoT Gateway could utilize a joint forecasting-scheduling architecture in which it predicts the future traffic of all IoT devices over a scheduling window and would schedule all of the traffic collision-free in advance. Thus determining to what extent *individual* IoT device traffic generation patterns are predictable is the first step towards understanding whether *proactive* solutions to the Massive Access Problem are viable.

In this chapter, we divide the IoT devices into four classes based on their traffic generation characteristics: (1) FBP, (2) VBP, (3) FBA, and (4) VBA. If an IoT device generates a constant number of bits over time in each generation instance, we categorize this device as “Fixed-Bit”; otherwise, we say that the device is “Variable-Bit”. Furthermore, if the device generates traffic at regularly spaced time instances, then we call the device “Periodic”; otherwise, we say that the device is “Aperiodic”.

Based on this categorization, in order to understand the predictability of IoT device traffic, we undertake a comparative study of four fundamental forecasting models for the problem of forecasting the traffic generation pattern of individual IoT devices in M2M communication. These forecasting models are: (1) ARIMA, (2) MLP, (3) 1-D CNN, and (4) LSTM.

In this chapter, by fine-tuning the architecture and the parameters of each of these four models, we measure the best performance attainable by each model in the sMAPE

[1]The technical content in this chapter has been published as a conference paper (Nakip, Gül, Rodoplu, & Güzeliş, 2019) in the Proceedings of the Cloud Computing and Internet of Things (CCIoT) 2019 Conference.

metric. We show that LSTM outperforms all of the other models significantly for devices in the VBP class in our simulations. Furthermore, we show that LSTM has almost the same performance in the sMAPE metric as MLP and 1D CNN for devices in the FBA class in our simulations. Finally, we show that both the training and execution times per IoT device of each of these four models are reasonable and can lead to a practical implementation at an IoT Gateway that employs joint forecasting-scheduling.

The rest of this chapter is organized as follows: In Section 2.2, we describe our forecasting models. In Section 2.3, we present our methodology and our key findings in regard to the forecasting performance as well as the training and execution times of our models. In Section 2.4, we present our conclusions and directions for future work.

2.2 Forecasting Models

Each forecasting model is characterized by its sequence of inputs, sequence of outputs, and the internal structure. The sequence of inputs and the sequence of outputs are common for all of the forecasting models, except ARIMA. Let recall the common notation for all models: At each discrete time slot m, the input of the forecaster of device i is subset $\{x_i[m-s^i_f]\}_{f\in F_i}$ of the past traffic generation pattern $\{x_i[m]\}$. In this notation, s^i_f is the time index, relative to the current time, that corresponds to the f^{th} feature of the traffic generation pattern of device i. Furthermore, F_i is the set of all indices of the features that are selected for device i.

The output of each forecaster, denoted by $\{\hat{x}_i[m+k]\}_{k\in\{1,\ldots,K_i\}}$, is the K_i-step ahead forecast for device i at current time m. The kth output corresponds to the value of the k-step ahead prediction, where $1 \leq k \leq K$. There are four possible ways in which to design a forecaster that will forecast the same output set $\{\hat{x}_i[m+k]\}_{k\in\{1,\ldots,K_i\}}$. These ways of designing the output are (2) the recursive structure, (2) vectorial output of a dense layer, (3) a set of softmax layers at the output each of which has a categorical output, and (4) using a bank of K_i forecasters each of which performs 1-step ahead forecasting. In this study, the ARIMA model is implemented with a recurrent structure, which is a standard design of ARIMA; the other models (MLP, 1D CNN, and LSTM) are designed with the vectorial output, denoted by $[1:K_i]$. Since the recursive structure propagates the error from $(k-1)$-step to k-step ahead forecasting, the recursive structure is not selected for the MLP, 1D CNN and the LSTM models. The structure is not selected as the set of softmax layers at the output because the categorical approach is not suitable for our forecasting problem. In addition, a separate forecaster for each of the K_i outputs is not used because of the space- and time-complexity of such usage. For the implementation of each forecaster in the JFS system, the value of K_i is calculated as $K_i = \lceil \frac{T_{sch}}{T_i} \rceil$; however, in order to analyze the error propagation of the forecasters as a function of the number of forecasting steps, the performance of forecasters is presented for an increasing sequence of K_i's in this chapter.

In this chapter, in order to measure the performance of the forecasting models, we used sMAPE. The sMAPE quantifies the forecasting error of the model as a general measurement technique. Furthermore, it shows how far the forecast value of the number of bits is from to the actual value.

Let *nSamples* denote the number of samples; thus, the actual traffic generation pattern is $\{x_i[m]\}_{m \in \{1,\ldots,nSamples\}}$, and its forecast is $\{\hat{x}_i[m]\}_{m \in \{1,\ldots,nSamples\}}$. The calculation of the sMAPE metric is given as

$$\text{sMAPE} = \frac{1}{nSamples} \sum_{m=1}^{nSamples} \frac{|\hat{x}_i[m] - x_i[m]|}{(|\hat{x}_i[m]| + |x_i[m]|)/2} \delta_{\hat{x}_i[m] \neq x_i[m]} \tag{2.1}$$

The original calculation of sMAPE in (2.1) covers only 1-step ahead forecasting. The sMAPE for multi-step ahead forecasting is calculated as

$$\text{sMAPE} = \frac{1}{nSamples \times K_i} \sum_{m=1}^{nSamples} \sum_{k=1}^{K_i} \frac{|\hat{x}_i[m+k] - x_i[m+k]|}{(|\hat{x}_i[m+k]| + |x_i[m+k]|)/2} \delta_{\hat{x}_i[m+k] \neq x_i[m+k]} \tag{2.2}$$

Note that the sMAPE metric returns a result in the interval $[0\%, 200\%]$. In order to perform a fair comparison of the forecasting models, the parameters of the internal structure of each model are optimized by using the methods that are explained in Section 2.3.2 in order to minimize sMAPE.

2.2.1 Auto-Regressive Integrated Moving Average

The well-known ARIMA model is selected as the benchmark against which machine-learning-based forecasting schemes (MLP, 1D CNN, LSTM) are compared. In this section, the formation of the ARIMA model for each IoT device i is described. The set of parameters of the ARIMA model are Autoregression (p), the Integration (d), and the Moving Average (q) coefficients. First, for the ARIMA model, the sequence of inputs to the ARIMA model are taken to be the entire past traffic generation pattern of device i. That is, the set F_i is not used to select the features of the input set because the ARIMA model is able to capture the seasonal trends with its q coefficient or *MALags*, which is a vector of positive integer lags associated with the moving average (MA) coefficients. In this study, instead of the q parameter, *MALags* is used in order to extract the important features more accurately. Note that we implemented the *MALags* of the ARIMA model as the additive seasonal lags. Thus, the *MALags* of the ARIMA model is set to F_i for each device i. Second, we perform exhaustive search in order to determine the local-optimal p and d, where the search ranges of these parameters are denoted by $[z_a^p, z_b^p]$ and $[z_a^d, z_b^d]$, respectively. Here, z_a^p and z_a^d are the lower bounds and z_b^p and z_b^d are the upper bounds of the search ranges. (The numerical setting for each of these parameters will be given in Section 2.3.2.)

2.2.2 Multi-Layer Perceptron with Local Optimal Structure

The internal architectural parameters of MLP are the number of layers E, and the number of neurons n_e in each layer e such that $1 \leq e \leq E$. We set all of these parameters via the MLP-SSA, which we describe below. This algorithm finds a local optimal solution for the problem of minimizing sMAPE across the validation set.

Fig. 2.1 shows the pseudo-code of the MLP-SSA function, which returns a local optimal structure for the MLP. Among the list of inputs to this function are (1) $\tilde{E}$, which denotes the maximum number of successive layers added such that no improvement is observed in the sMAPE metric, and (2) $\tilde{N}$, which denotes the maximum number of neurons (for each layer) that are successively added without improvement in the sMAPE metric. On Line 2, a new MLP architecture is created that has n_0^i inputs[2] and K_i outputs. (The i index refers to the device for which the MLP architecture is formed.) On Lines 3-4, the lowest sMAPE value, denoted by $\text{sMAPE}_{\text{allLayers}}^{\text{lowest}}$, over all of the layers, is set to infinity, and the number of layers that are added without gaining any improvement, namely nLayersAddedWithoutImprovement, is set to zero. The main idea behind Lines 5-25 is as follows: The algorithm begins (Line 6) by adding an empty hidden layer (that does not yet have any neurons) to mlp. Then, for this hidden layer, it keeps adding neurons until for $\tilde{N}$ successive stages, there is no improvement in the sMAPE metric. If this stopping condition is satisfied, then the algorithm deletes (Line 18) the last $\tilde{N}$ neurons that it has added (since these have not resulted in any improvement). The algorithm keeps adding additional hidden layers as long as the number of layers added without improvement is less than $\tilde{E}$ (Line 5). If the stopping condition for this outer loop is

[2]In this chapter, we set the value of n_0^i to the length of F_i.

MLP Structure Selection Algorithm (MLP-SSA)

```
1    MLP_Arch*
         MLP_SSA($\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}, \mathbf{X}_{\text{val}}, \mathbf{Y}_{\text{val}}, n_0^i, K_i, \tilde{E}, \tilde{N}$) {
2      $mlp$ = new MLPArch($n_0^i, K_i$);
3      $\text{sMAPE}^{\text{lowest}}_{\text{allLayers}} = +\infty$;
4      nLayersAddedWithoutImprovement = 0;
5      while(nLayersAddedWithoutImprovement $< \tilde{E}$) {
6        AddEmptyHiddenLayer($mlp$);
7        $\text{sMAPE}^{\text{lowest}}_{\text{currentLayer}} = +\infty$;
8        nNeuronsAddedWithoutImprovement = 0;
9        while(nNeuronsAddedWithoutImprovement $< \tilde{N}$) {
10         AddNeuron($mlp$);
11         TrainMLP($mlp$, $\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}$);
12         sMAPE = TestMLP($mlp$, $\mathbf{X}_{\text{val}}, \mathbf{Y}_{\text{val}}$);
13         if($\text{sMAPE} \geq \text{sMAPE}^{\text{lowest}}_{\text{currentLayer}}$)
14           nNeuronsAddedWithoutImprovement++;
15         else
16           $\text{sMAPE}^{\text{lowest}}_{\text{currentLayer}}$ = sMAPE;
17       }
18       DeleteLastNeurons($mlp, \tilde{N}$);
19       if($\text{sMAPE}^{\text{lowest}}_{\text{currentLayer}} \geq \text{sMAPE}^{\text{lowest}}_{\text{allLayers}}$)
20         nLayersAddedWithoutImprovement++;
21       else
22         $\text{sMAPE}^{\text{lowest}}_{\text{allLayers}} = \text{sMAPE}^{\text{lowest}}_{\text{currentLayer}}$;
23     }
24     DeleteLastLayers($mlp, \tilde{E}$);
25     return $mlp$;
26   }
```

Figure 2.1. Pseudo-code for the MLP Structure Selection Algorithm (MLP-SSA) that is used for the selection of the local optimal MLP architecture

satisfied, the algorithm deletes (Line 24) the last $\tilde{E}$ layers that it has added and returns the resulting MLP (Line 25).

2.2.3 1-Dimensional Convolutional Neural Network

In this section, we design a 1D CNN, which is shown for device i in Fig 2.2. We experimentally selected the internal architecture of the 1D CNN model as follows: a convolution layer, a max pooling layer, global average pooling, dropout, and a fully connected layer. We use global average pooling to avoid overfitting by reducing the total number of parameters in the model. Furthermore, we set the Fully Connected Layer in Fig. 2.2 as the MLP found via the MLP-SSA algorithm described in Section 2.2.2.

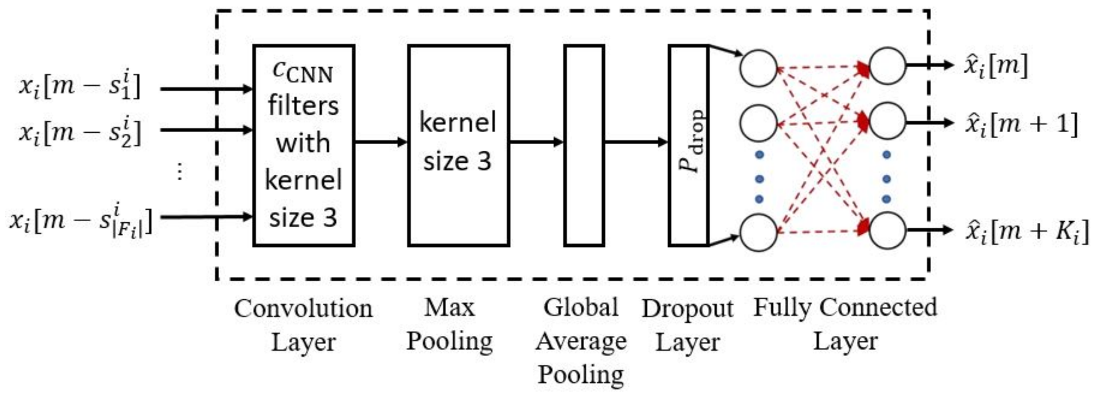

Figure 2.2. 1D CNN for $[1 : K_i]$-step ahead forecasting

We set the kernel size of the convolution layer and the max pooling layer to 3. We also set the stride of the convolution layer to 2. We empirically selected the activation function of the convolution layer and that of each neuron at each of the first three hidden layers as $ReLU$, and we did not use any activation function for any neuron at the output layer. Note that the number of neurons at the output layer is equal to K_i. The dropout probability p_{drop} is set to 0.15. We search the range $[z_a^{CNN}, z_b^{CNN}]$ in order to find the local-optimal number of filters c_{CNN} of the convolution layer.

2.2.4 Long-Short Term Memory Model

In this section, we use the LSTM model to forecast the IoT data traffic. We design the structure of the LSTM model with the following sequence of layers after the input layer: the lstm layer, two fully connected layers (indexed by $\bar{e} = 1$ and $\bar{e} = 2$), a dropout layer, and the output layer, which is a fully connected layer with index $\bar{e} = 3$. In this sequence, we use the dropout layer in order to increase the generalization ability of the model, and we set the dropout probability to 0.15. We denote by h_{lstm} the number of lstm units in the lstm layer. Each lstm unit includes an update gate, a forget gate and an output gate. Furthermore, $h_{\bar{e}}$ denotes the number of neurons in each fully connected layer $\bar{e} \in \{1, 2, 3\}$. Since K_i-step ahead forecasting requires K_i outputs, we set $h_3 = K_i$. The output of the k^{th} neuron in the output layer is the k^{th}-step ahead forecast. We take the dropout probability p_{drop} to be equal for all devices and for all values of K_i. In order to find the values of the parameters h_{lstm}, h_1 and h_2, we search the ranges $[z_a^{lstm}, z_b^{lstm}]$, $[z_a^{h_1}, z_b^{h_1}]$ and $[z_a^{h_2}, z_b^{h_2}]$, respectively.

2.3 Results

2.3.1 IoT Dataset

In this section, our goal is to select representative devices from each of the three IoT device classes (VBP, FBA and VBA) for which forecasting is required. (Note that no forecasting is required for the FBP class since each device in this class generates a fixed number of bits at regular intervals; hence, its traffic generation pattern is completely predictable.) In actual IoT systems, if the data from a sensor connected to the IoT device is sent without compression, the data takes a fixed number of bits. Furthermore, if the data is sent in an event-triggered fashion, where the IoT device transmits the new sensor data only when a significant change has occurred in the value of the sensor reading compared with the previous measurement, then the generated device traffic is aperiodic; that is, the traffic is not generated at regular time intervals. We apply these facts in generating traffic from IoT devices as follows: First, in our laboratory, we collected data from the Relative Humidity (RH), Non-Methane Hydrocarbon (NMHC), and Nitrogen Dioxide (NO_2) sensors. Second, by using Huffman coding on the collected sensor data from the RH and the NO_2 sensors, we obtained a traffic generation pattern for these devices that are variable-bit. Furthermore, assuming that the devices connected to the NMHC and NO_2 sensors transmit their data in an event-triggered fashion, we obtained a traffic generation for these devices that are aperiodic. Thus, under these assumptions, the traffic generation pattern of the RH sensor falls in the VBP class, the NMHC in the FBA class, and NO_2 in the VBA class.

2.3.2 Parameter Tuning and Training

We trained and tuned the ARIMA and MLP models on MATLAB, and the 1D CNN and LSTM models in Python on the Google Colab platform using the Tensor Processing Unit (TPU) accelerator. Furthermore, in all of the models below, for each device i, we obtained the results for $[1 : K_i]$ when K_i takes values in the sequence $\mathbf{K} \equiv [1,2,3,4,5,10,15,20,25,30]$.

ARIMA:

In order to train the ARIMA model, we used maximum likelihood estimation. We implemented the ARIMA model by using the Econometrics Toolbox in MATLAB.

The search interval for the parameter p is $[z_a^p, z_b^p] = [0,3]$ and for the parameter d is $[z_a^d, z_b^d] = [0,3]$. The local optimal values of the parameters of the ARIMA model were found via exhaustive search for the parameters to be $(p,d,q) = (0,1,24)$ for the RH sensor, $(0,1,1)$ for the NMHC sensor, and $(0,1,3)$ for the NO_2 sensor. We note that the local optimal value of the Autoregression coefficient (p) of the ARIMA model is 0 for each of these devices.

MLP:

The connection weights and the biases are learned in the training stage by using the Levenberg-Marquardt (LM) algorithm in order to minimize the Mean Square Error (MSE). In order to forecast the traffic of all IoT devices in the VBP, FBA and VBA classes, we use a variable learning rate in the LM algorithm. We set the initial learning rate to 0.001. Furthermore, we set the decrease and increase factors to 0.1 and 10, respectively, and the maximum learning rate to 10^{10}. All of the hidden-layer neurons utilize the *tanh* activation function while the output neurons are kept linear.

The resulting neural network architecture for each IoT device is given in Table 2.1. Each entry in this table is indexed by the vector $[1 : K_i]$ for vector K_i-step ahead prediction (which appears at the top of each column) and the particular device (which appears on the left hand side of each row). In the table, we further show the class (VBP, FBA, VBA) in which each device falls. Each cell of Table 2.1 represents the local optimal MLP architecture that has been found for that device and for that $[1 : K_i]$. The local optimal MLP architecture is in the form $[n_1\ n_2\ \cdots\ n_E]$, where E refers the total number of layers. We see that for the majority of the cells in Table 2.1, the local optimal MLP architecture for the RH dataset has a number of layers that is greater than or equal to those for the other two sensor devices. The reason is that the traffic generation pattern of a device in the VBP class has uncertainty only in the number of bits rather than the timing of traffic generation; as a result, it tends to be more predictable than those for the FBA and the VBA classes.

We trained the MLP model in the batch mode. We selected the optimal number of epochs via early stopping, which stops the training when the MSE of the validation set does not decrease for 50 successive epochs. Below, we give the sequence of the optimal number of epochs that corresponds to the sequence of values in **K** for each device class.

- RH (VBP class): 14, 11, 22, 18, 7, 8, 12, 10, 41, 26
- NMHC (FBA class): 6, 13, 54, 14, 19, 14, 9, 10, 13, 11

Table 2.1. Local optimal network architectures of MLP found by the MLP-SSA algorithm up to 30-step ahead prediction for each IoT device

Device	Class	[1]	[1:2]	[1:3]	[1:4]	[1:5]
RH	VBP	[2 6 5 3]	[1 3 5 8]	[1 3]	[1 3 9 6 4]	[5]
NMHC	FBA	[6 1 1 1]	[1 1]	[1]	[2 1]	[1]
NO_2	VBA	[6]	[5 6 11]	[6 9]	[6 7 13]	[5 13 8]

Device	Class	[1:10]	[1:15]	[1:20]	[1:25]	[1:30]
RH	VBP	[4 12 3 6]	[2 5 8 8]	[3 9]	[2 3]	[2]
NMHC	FBA	[1]	[2]	[2]	[2]	[2]
NO_2	VBA	[6 7]	[6 6]	[6 6 7]	[5 6]	[5 7 8]

- NO_2 (VBA class): 10, 16, 17, 16, 14, 15, 10, 15, 9, 13

1D CNN:

We set the activation function of each neuron of the 1D CNN as follows: We use *ReLU* for the convolution layer, *tanh* for the hidden layers of the Fully Connected Layer, and no activation function (resulting in a *linear* structure) for the output layer. In order to find the number of filters c_{CNN} in the convolution layer, we set the search interval as $[z_a^{CNN}, z_b^{CNN}] = [8, 256]$. We measure the value of sMAPE of the test set for each value in this search interval that is in the form 2^m for integer m: $3 \leq m \leq 8$.

For the sequence of values in **K**, the following are the number of filters: 256, 128, 8, 32, 64, 64, 128, 32, 256, 256 for the RH sensor. For the NMHC sensor, the sMAPE error turned out to be identical (around 24%) for all values of c_{CNN}. Thus, we took the number of filters to be 8 for the sequence of values in **K** in order to keep the model simple. We found the number of filters for the NO_2 sensor to be 256, 16, 32, 64, 32, 8, 8, 8, 8, 8 for the sequence of values in **K**. For the majority of values in **K**, we see that the 1D CNN model requires more filters for the most predictable dataset, namely for RH.

The best epoch is determined by using early stopping whose stopping criterion is identical to the one used for MLP. The best epochs for the sequence of values in **K** are as follows:

- VBP: 87, 77, 163, 131, 114, 119, 87, 146, 103, 119
- FBA: 101, 56, 58, 55, 58, 70, 66, 64, 70, 64
- VBA: 235, 169, 174, 206, 215, 254, 262, 194, 285, 198

LSTM:

For the training of the LSTM model, we used Adaptive moment estimation (*Adam*), which is a first-order gradient-based optimization algorithm, in order to minimize the MSE of forecasting. We implemented the same early stopping technique as for MLP and 1D CNN. The best epochs for the sequence of values in $\mathbf{K}$ are as follows:

- VBP: 96, 67, 76, 144, 67, 75, 62, 172, 65, 92
- FBA: 93, 80, 69, 85, 129, 155, 168, 147, 54, 128
- VBA: 243, 248, 213, 204, 70, 69, 86, 55, 60, 56

The search intervals for the parameters h_{lstm}, h_1 and h_2 are as follows: $[z_a^{lstm}, z_b^{lstm}] = [8, 256]$; $[z_a^{h_1}, z_b^{h_1}] = [32, 128]$ and $[z_a^{h_2}, z_b^{h_2}] = [16, 64]$. In each interval, we take only the values that have the form 2^m for $m : 3 \leq m \leq 8$. The resulting parameters of the LSTM model are shown in Table 2.2 for each device. Each cell of Table 2.2 is in the form (h_{lstm}, h_1, h_2). Table 2.2 shows that in the majority of cases, h_{lstm} is higher for the VBA class than for the VBP and the FBA classes.

Table 2.2. Parameters of LSTM found up to 30-step ahead prediction for each IoT device

Device	Class	[1]	[1:2]	[1:3]	[1:4]	[1:5]
RH	VBP	(8, 128, 32)	(32, 128 16)	(256, 64, 64)	(8, 32, 16)	(16, 32, 64)
NMHC	FBA	(8, 32, 16)	(8, 32, 32)	(8, 64, 16)	(16, 32, 16)	(8, 64, 64)
NO_2	VBA	(32, 128, 32)	(32, 32, 32)	(64, 32, 16)	(32, 128, 64)	(256, 64, 16)

Device	Class	[1:10]	[1:15]	[1:20]	[1:25]	[1:30]
RH	VBP	(8, 64, 64)	(8, 128, 64)	(8, 32, 32)	(16, 32, 32)	(16, 128, 32)
NMHC	FBA	(8,64,16)	(8, 32, 16)	(8, 32, 16)	(256, 32, 64)	(8, 32, 64)
NO_2	VBA	(64, 32, 16)	(256, 64, 64)	(32, 128, 32)	(64, 128, 16)	(8, 32, 64)

2.3.3 Comparison of Forecasting Performance Results

In this section, we present the performance of forecasting schemes, measured via sMAPE. First, our goal is to validate that the traffic generation patterns of the actual devices from which we collected data are representative of those for the VBP, FBA and VBA device classes. To this end, we formed bootstrapped traffic generation patterns from those of the actual devices while retaining inter-temporal statistics of the traffic generation patterns. We trained each such bootstrapped device separately while preserving the same architecture as for the original device. Then, we compared the forecasting performance for these devices with those for the actual devices. We validated that the results presented in this section are similar to those for bootstrapped devices in that the sMAPE displayed a standard deviation of approximately 10% of the mean in each case. Since bootstrapping preserves the statistics of the original traffic generation patterns while generating new realizations, our tests suggest that the forecasting results presented in this section are robust to such variations.

We note that the sMAPE metric takes values in the range [0%, 200%]. We follow the original definition of sMAPE in this regard, for which the metric gives 200% when all of the forecasting results have maximal error, and gives 100%, when half of the results are in error in a binary classification problem (e.g. as for the FBA class).

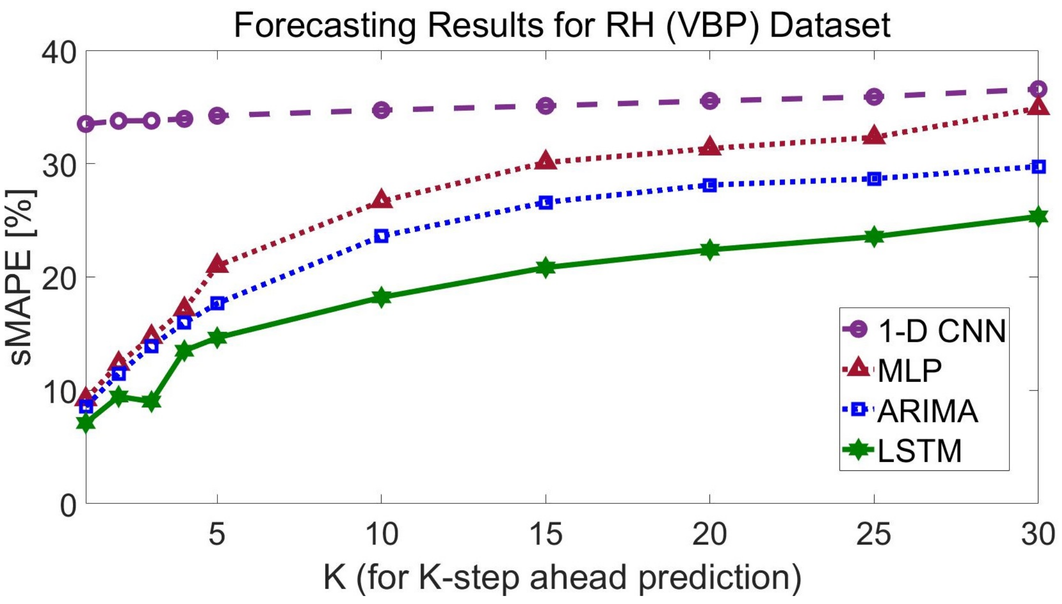

Figure 2.3. sMAPE results of RH (VBP) traffic generation pattern up to 30-step ahead prediction

In Fig. 2.3, we show the forecasting error (sMAPE) for the traffic generation pattern of the RH sensor, which falls in the VBP class. First, we see that LSTM outperforms the other three models. The reason is that the traffic generation pattern of the RH sensor has seasonal trends and LSTM captures both the short and the long term seasonal trends in the generation pattern. Since 1D CNN, MLP and ARIMA do not have forget and update gates in their structure, they do not capture these seasonal trends as effectively as LSTM.

Fig. 2.4 shows the forecasting error for the NMHC sensor, whose traffic generation pattern falls in the FBA class. We see that LSTM, 1D CNN, MLP outperform the

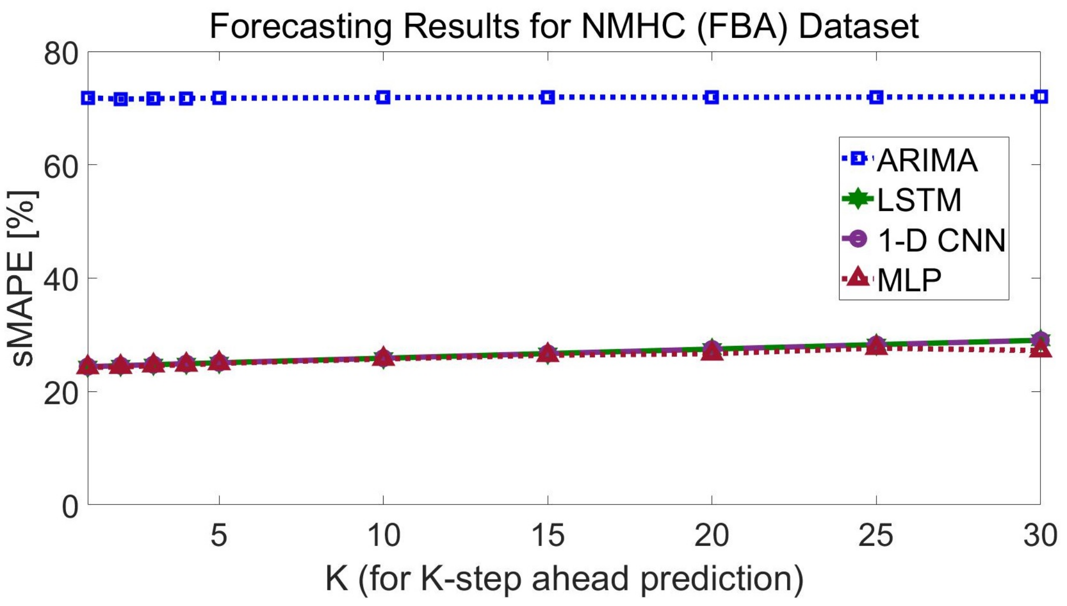

Figure 2.4. sMAPE results of NMHC (FBA) traffic generation pattern up to 30-step ahead prediction

ARIMA model by a significant margin in the sMAPE metric. We note that forecasting the traffic generation pattern for the FBA class is equivalent to a binary classification problem in which we represent the traffic at each discrete time instance by 0 or the fixed number of bits that the device can send. The reason for the much worse performance of ARIMA in Fig. 2.4 is that as a linear model, ARIMA performs well in those cases in which it captures variables that vary continuously as a function of time. As a result, ARIMA underperforms in capturing the non-linear relationship implied by the binary classification problem.

In Fig. 2.5, first, we see that none of the four models are able to perform sufficiently accurate forecasts of the traffic generation pattern of the NO_2 sensor, which falls in the VBA class. The reason is that, compared with the VBP and the FBA classes, both the number of bits and the traffic generation times must be successfully predicted for the VBA class. Second, we see that ARIMA outperforms the other three models in this case. The reason is as follows: Since the ARIMA model is linear and the Autoregression coefficient (p) of the model is 0 in our local optimal setting, the generalization ability of ARIMA is expected to be higher than those of the other three models. As a result, when all of the models have high sMAPE, ARIMA outperforms 1D CNN, LSTM and MLP, all of which are non-linear models.

Our main conclusion is that LSTM appears as the most promising model out of the four models in regard to the forecasting performance, as measured by sMAPE. However,

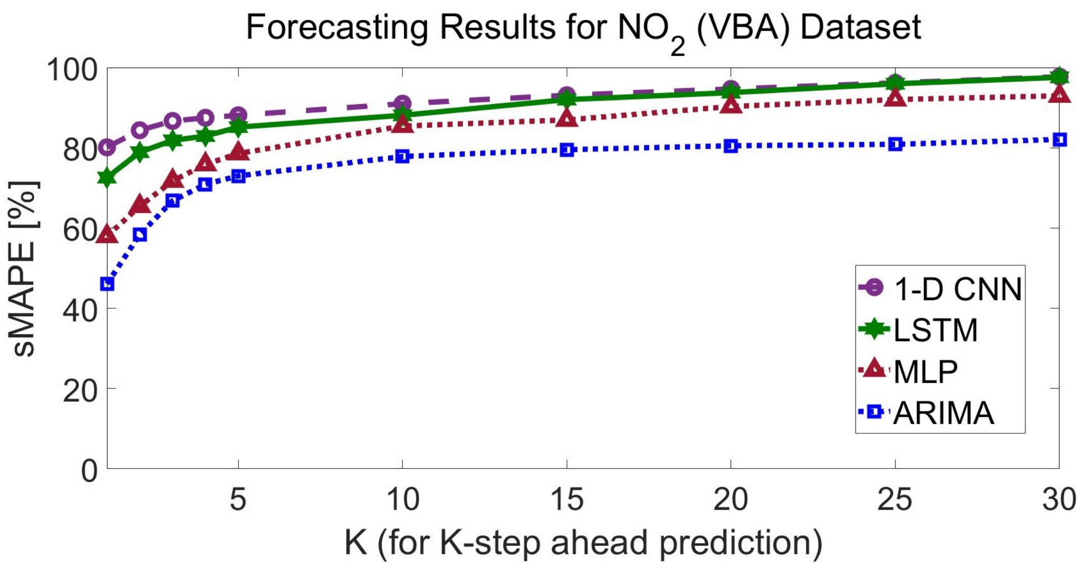

Figure 2.5. sMAPE results of NO_2 (VBA) traffic generation pattern up to 30-step ahead prediction

this performance must be compared against the training and execution times of this model in order to determine whether the model can be implemented in practice. In the next section, we undertake an examination of the training and execution times of these models.

2.3.4 Comparison of Models with Respect to Training Time

We first examined the execution times of all of the four models and found that all execution times are less than 0.5 seconds on a 4.00 GHz DL380 Gen10 Intel Xenon-Gold 6138 48-Core Processor with 128 GB of RAM for the ARIMA and MLP models and on the Google Colab platform for the 1D CNN and LSTM models. This execution time is sufficient for most of the IoT applications that utilize sensors. In the rest of this section, we focus on a comparison of the training times of the four models. The training times that we report in this section have been obtained as the average over 10 training trials for each of the devices for each forecasting model.

In Fig. 2.6, for the RH sensor (which falls in the VBP class), we see that all of the four models have training times that are less than 150 seconds. This training time is reasonable, assuming that the models will be trained off-line. (For online training, further improvements to the models would be necessary, which we plan to undertake in our future work.) Second, while ARIMA has a training time that is approximately 0.3 seconds, the training time of LSTM is approximately 50 to 130 seconds. This

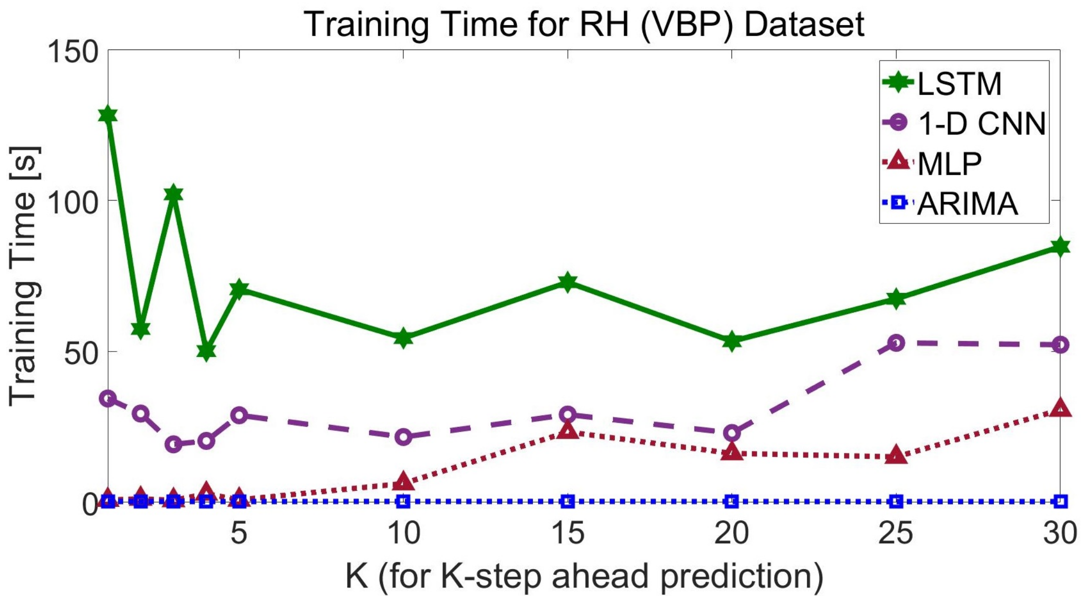

Figure 2.6. Training time of RH (VBP) traffic generation pattern up to 30-step ahead prediction

300-fold difference in the training times of the two models must be compared against the forecasting performance achieved by each model. Since the sMAPE of LSTM is lower than that of ARIMA by at most 5%, network designers must carefully decide how to trade off the forecasting performance against the training time in order to decide between LSTM and ARIMA.

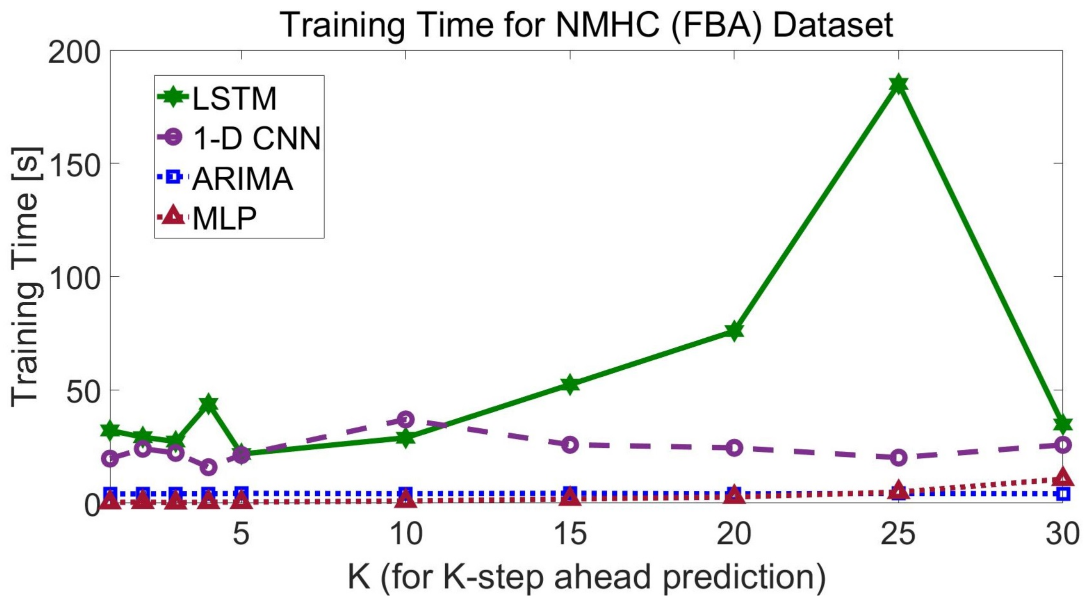

Figure 2.7. Training time of NMHC (FBA) traffic generation pattern up to 30-step ahead prediction

In Fig. 2.7, for the NMHC sensor (which falls in the FBA class), first, we see that LSTM has a training time that is significantly larger than those of the other models beyond $K = 15$ for the FBA class. In particular, the LSTM training time peaks at $K = 25$. The reason is that the number of lstm units h_{lstm} (shown in Table 2.2) equals 256 for $K = 25$, which is much higher than those for the other values of K for LSTM. Second, the training times of MLP and ARIMA are much lower than those of 1-D CNN and LSTM. The reason is that the local optimal structures of MLP (shown in Table 2.1) and ARIMA model are simpler than the structures of LSTM and 1-D CNN. Taking into account both the training time and the sMAPE performance (Section 2.3.3), we see that MLP outperforms all of the other models in regard to both of these metrics in this case.

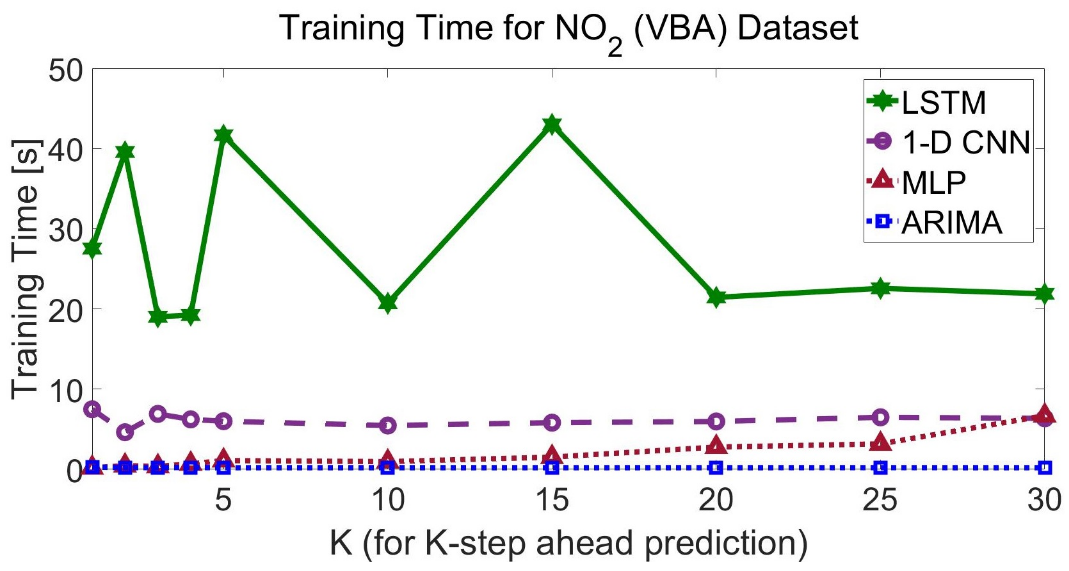

Figure 2.8. Training time of NO_2 (VBA) traffic generation pattern up to 30-step ahead prediction

In Fig. 2.8, for the NO_2 sensor (which falls in the VBA class), first, we see that the training time of LSTM is much higher than those of the other three models. Second, we see that the training time of ARIMA is the lowest of all of the four models, since it has the simplest structure out of the four models. We note that ARIMA outperforms all of the other models in both the training time and the forecasting performance in this case. Finally, we note that the reason that the training time of the LSTM model peaks at $K = 5$ and $K = 15$ is that h_{lstm} equals 256 in these cases, which is much higher than that for the other values of K.

2.4 Summary

In this chapter, we showed that to the extent that the devices in our simulations are representative of the VBP, FBA and the VBA classes, the traffic generation patterns in the VBP and the FBA classes appear far more predictable than that in the VBA class using the best forecasting model in each case. We also found that the LSTM model outperforms all of the other models for the VBP class. Furthermore, we demonstrated that for the FBA class, the performance of LSTM, 1D CNN and MLP are comparable and much better than that of ARIMA. For the VBA class, none of the four models achieve a sufficiently high forecasting performance for K-step ahead prediction for $K > 5$. In regard to training time, LSTM has a significantly higher training time than the other three models. Furthermore, ARIMA has the lowest training time out of the four models. When the forecasting error and the training time are examined jointly, for the VBP class, LSTM and ARIMA are comparable; for the FBA class, MLP outperforms all of the other models, and for the VBA class, ARIMA outperforms all of the other models.

Our results suggest a joint forecasting-scheduling architecture to be implemented at an IoT Gateway in which the Gateway predicts the future traffic of devices in distinct device classes and schedules their future traffic jointly based on these forecasts. This chapter constitutes a first step towards building predictive networks, in which the network predicts the traffic generation patterns of individual devices. Such predictions can be subsequently used for efficient allocation of network resources based on forecasts. This, in turn, holds the potential to deliver a much higher network performance than can be achieved via reactive schemes that merely respond to current traffic demand.

Chapter 3
JOINT FORECASTING-SCHEDULING SYSTEM

3.1 Introduction

As we have discussed in Chapter 1, the Massive Access Problem of the IoT refers to the problem of enabling the wireless access of a massive number of IoT devices to the wired Internet. Among the diverse set of IoT devices, smart utility meters alone are expected to reach a density of approximately 5,000 to 35,000 devices per base station in urban areas (Vodafone, Apr. 2010). Since the 4G wireless systems will be unable to handle such an enormous number of devices per base station due to the PRACH overload (Ghavimi & Chen, 2015), novel access solutions must be devised for the IoT revolution to take off.

The past work on the Massive Access Problem (Aijaz et al., 2014) (Aijaz & Aghvami, 2015)(Jin et al., 2017)(Liang et al., 2018)(Lien et al., 2012)(Lin et al., 2014) has modeled the traffic generation patterns of IoT devices via "random arrivals" at the MAC layer. Under the random arrival model, since the future traffic of IoT devices cannot be predicted accurately, only MAC-layer protocols that are *reactive* to the traffic demand are feasible. In contrast, if the future traffic of individual IoT devices can be predicted accurately, then *proactive* scheduling of device traffic becomes viable. Proactive scheduling has the advantage of eliminating contention, collision and handshaking when implemented centrally by an IoT Gateway or base station.

Our results in Chapter 2 indicate that the traffic generation patterns of certain classes of *individual* IoT devices for M2M communication can be predicted with relatively high accuracy in the sMAPE metric via machine and deep learning techniques. However, in that chapter, the implications of the predictability of the traffic generation patterns of individual IoT devices for scheduling were not investigated.

The first contribution of this chapter[1] is the design of a JFS system, to be implemented

[1]The technical content in this chapter has been published as a conference paper (Nakip, Rodoplu, Güzeliş, & Eliiyi, 2019) at the IEEE Global Conference on the Internet of Things (GCIoT) 2019.

at an IoT Gateway, that forecasts the future traffic generation pattern of each IoT device and schedules the traffic of these devices in advance without any collisions over a scheduling window. Not only does such a system have the advantage of potentially achieving a high throughput compared with reactive solutions to the Massive Access Problem, but it can also minimize the average transmit energy spent by each IoT device and thus prolong the lifetime of battery-limited IoT devices.

The second contribution of this chapter is a comparison of the performance of JFS with respect to the forecasting scheme employed. In particular, we evaluate the network performance of ARIMA against the machine-learning based MLP and LSTM models when each of these models is used for forecasting the future traffic generation patterns of IoT devices. In this chapter, we measure network performance via the uplink cross-layer throughput in a full JFS system. This provides a much more accurate measure of performance than traditional performance metrics for forecasting (such as MSE or sMAPE) that do not directly quantify the impact of forecasting on scheduling.

The third contribution of this chapter is the development of a reservation-based heuristic scheduling algorithm, which we call Priority based on Average Load (PAL), that allocates the MAC-layer slots for IoT devices. The PAL achieves high throughput performance in the majority of cases. Furthermore, its time complexity is linear in the number of devices. Via our simulations, in this chapter, we estimate the computation time that would be required to implement JFS in an actual system that operates at an IoT Gateway.

The rest of this chapter is organized as follows: In Section 3.2, we present our system for joint forecasting-scheduling. In Section 3.3, we present our results on the network performance and the computation time of our system. In Section 3.4, we present our conclusions.

3.2 Joint Forecasting-Scheduling System

The main idea behind JFS is as follows: The Gateway G forecasts the traffic generation pattern $\{x_i[m]\}$ of each device i over a scheduling window of duration T_{sch}. The Gateway then aims to schedule the devices' uplink transmissions collision-free over this window based on its forecasts. On a downlink broadcast channel, G communicates to each device i the sequence of slots that are allocated to i on the uplink. In our system, on the downlink, G communicates to each i this sequence of slots for i's *next forecast burst*.

Whenever each device i transmits to G, it prepends to its transmission a compressed representation of its actual generated traffic pattern $\{x_i[m]\}$ since its last transmission. Based on this information, G forecasts the *future* traffic pattern $\{\hat{x}_i[m]\}$ of that device.

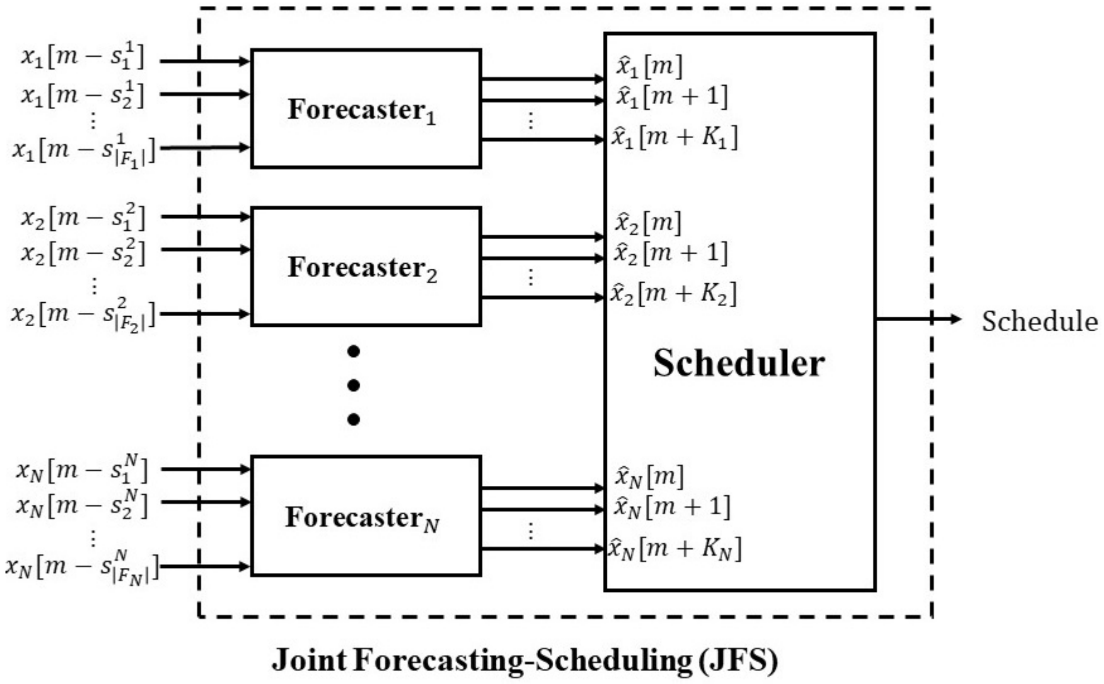

Figure 3.1. Joint Forecasting-Scheduling (JFS) System

Fig. 3.1 displays our JFS system. Note that there is a single forecaster for each individual IoT device. In contrast, scheduling is performed jointly for all IoT devices over the scheduling window of duration T_{sch} based on the outputs of the bank of forecasters. In Fig. 3.1, each forecaster operates at a potentially different rate from those of the other forecasters. In particular, the forecaster for device i operates at rate $1/T_i$, where T_i is the period at which device i generates its traffic. The scheduler takes in the forecasts from the bank of forecasters and forms a schedule of duration T_{sch} that is represented at the granularity of a MAC-layer slot.

We assume that we perform feature selection as a preprocessing step before each forecaster. In Fig. 3.1, the input of each forecaster is a subset $\{x_i[m - s_f^i]\}_{f \in F_i}$ of the past traffic generation pattern $\{x_i[m]\}$ of device i. In this notation, s_f^i is the time index, relative to the current time, that corresponds to the f^{th} feature of the traffic generation pattern of device i. Furthermore, F_i is the set of all of the indices of the features selected for device i. For example, if only the last five past samples of the traffic generation pattern of device i are selected as features, $F_i = \{1, 2, 3, 4, 5\}$. The output of each forecaster, denoted by $\{\hat{x}_i[m + k]\}_{k \in \{1,\dots,K_i\}}$, is the k^{th}-step ahead forecast for device i at current time m over the next K_i samples. Note that $K_i = \lceil \frac{T_{sch}}{T_i} \rceil$.

3.2.1 Forecasting of IoT Traffic

In this chapter, we shall present a comparison of the performance of three different forecasting models (ARIMA, MLP, and LSTM) with respect to the network throughput of JFS. To this end, in this section, we discuss the parameters of each of these forecasting models. Note that since the 1D CNN model underperforms the other models as shown in Section 2.3.3, we do not use this forecasting scheme within the JFS system.

Assume that we fix the forecasting model for each forecaster in the bank of forecasters as ARIMA, MLP or LSTM. Then, as a metric for training and tuning the bank of forecasters, we shall focus on the throughput difference Φ between the throughput η_o obtained under perfect forecasts and the throughput η obtained under our forecasting scheme over a scheduling window of duration T_{sch}. The equation of Φ is given by

$$\Phi = \eta_o - \eta \tag{3.1}$$

Each forecasting model is defined by its sequence of inputs, its sequence of outputs and its internal structure. We determine the architectural parameters of the internal structure of each model by minimizing Φ.

We shall now describe the internal structure of each of the forecasting models.

Auto-Regressive Integrated Moving Average (ARIMA):

We now explain how to form the ARIMA model for each IoT device i. ARIMA takes all of the past samples as input for each device i and selects the set F_i based on *MALags*, which is a vector of positive integer lags associated with the MA coefficients. (Note that we do not use the MA coefficient q.) The other parameters of ARIMA are the autoregression coefficient p and the integration coefficient d. The range of values over which we shall search for the local optimal value of p shall be denoted by $[z_a^p, z_b^p]$, where z_a^p is the lower bound and z_b^p is the upper bound of the search range. Similarly, the search range for d shall be denoted by $[z_a^d, z_b^d]$. After selecting the set F_i, we perform exhaustive search simultaneously for p and d over a two-dimensional grid that consists of their respective search ranges.

Multi-Layer Perceptron (MLP):

The internal architectural parameters of the MLP are the number of layers E and the number of neurons n_e at layer $e \in \{1, \ldots, E\}$. We determine these parameters via a Neural Network Selection Algorithm that we have devised. This algorithm finds a local optimal value of Φ and is denoted by MLP-SSA-Φ.

The MLP-SSA (which is given in Fig. 2.1 in Section 2.2.2) is revised by replacing the sMAPE calculation with Φ in order to design MLP-SSA-Φ. The MLP-SSA-Φ algorithm works as follows: The algorithm successively adds empty layers, starting with the formation of the input layer. At each layer that has been added, the algorithm increments the number of neurons at that layer (starting with a single neuron) until the value of Φ (which is obtained at the output of scheduling) no longer decreases. In this case, the algorithm deletes the last neuron that was added that resulted in an increase in Φ and fixes the number of neurons at that layer. The algorithm keeps track of the smallest value of Φ, denoted by Φ_{min}, that has been achieved thus far in the process of adding layers. If Φ_{min} does not decrease upon the addition of $\tilde{E}$ successive layers (where $\tilde{E}$ is a parameter to be chosen), then MLP-SSA-Φ deletes the last $\tilde{E}$ layers that have been added and returns the resulting architecture as the final architecture of the MLP model.

Long-Short Term Memory (LSTM):

We now explain how we determine the structure of the LSTM model. We form the LSTM model with a single lstm layer, two fully connected layers (indexed as $\bar{e} = 1$ and $\bar{e} = 2$), a dropout layer and the output layer, which is a fully connected layer indexed as $\bar{e} = 3$. We tune the following architectural parameters of the LSTM model: (1) the number of lstm units h_{lstm} in the lstm layer, and (2) the number of neurons $h_{\bar{e}}$ in each fully connected layer $\bar{e} \in \{1, 2, 3\}$. Note that each lstm unit consists of the following gates: an input, an output, an update and a forget gate. We set the value of the number of neurons in the output layer, namely h_3, to K_i for device i. Furthermore, we search for the local optimal values of the parameters h_{lstm}, h_1 and h_2 in the search intervals $[z_a^{h_{lstm}}, z_b^{h_{lstm}}]$, $[z_a^{h_1}, z_b^{h_1}]$ and $[z_a^{h_2}, z_b^{h_2}]$, respectively.

3.2.2 Scheduling

We now explain how we perform scheduling for our JFS system, which was shown in Fig. 3.1. Each burst j is generated in time slot r_j and must be delivered by time slot d_j. (Thus, $\Delta_j = d_j - r_j$.) We let a_j denote the total number of bits in burst j. The processing time p_j of burst j is defined as the total number of MAC-layer slots that the transmitting IoT device needs in order to send burst j to Gateway G. (As stated in Section 1.4, JFS schedules each transmission in a distinct time slot such that it does not collide with any other transmission.) We let R_j denote the data rate (in bits per second) at which burst j can be transmitted to G. Furthermore, we let τ_{MAC} denote the duration (in seconds) of each MAC-layer slot. Thus, $p_j = \lceil \frac{a_j}{R_j \tau_{MAC}} \rceil$.

While scheduling may focus on a variety of objective functions, in this chapter, we focus on the uplink cross-layer throughput η (as defined in Section 1.4). Furthermore, for simplicity, we develop a heuristic, called PAL, and use it as the scheduling algorithm in this chapter. (However, we emphasize that JFS is a general methodology, and a variety of scheduling algorithms may be used in the Scheduler module of JFS in Fig. 3.1.)

Let $\mathscr{J}$ denote the set of all bursts across all devices $\mathscr{N}$ over the current scheduling window. Let $\tilde{a}_j[m]$ denote the number of bits of burst j that have not yet been sent to G. Furthermore, we let $\tilde{\Delta}_j[m]$ denote the length of time that remains at the current time m until the delay deadline d_j of burst j. We let $\mathscr{J}_{active}[m]$ denote the set of "active bursts", namely, those bursts which have been generated but not yet completely processed and whose delay deadlines have not yet expired at the current time m. Then, we define the "effective load" $\gamma_j[m] \equiv \frac{\tilde{a}_j[m]}{\tilde{\Delta}_j[m]}$ for each $j \in \mathscr{J}_{active}[m]$ at the current discrete time m.

Now, PAL is a greedy algorithm that schedules bursts on a single channel in a non-preemptive fashion. "Non-preemptive scheduling" means that if an IoT device has started sending a burst, the algorithm continues to send that burst until the transmission of the burst has been completed. That is, no other burst from any of the IoT devices can be scheduled on the single channel while the current burst is being transmitted. After the transmission of the current burst has been completed, PAL computes $j^* = \arg\max_{j \in \mathscr{J}_{active}[m]} \gamma_j[m]$ for the current time m; that is, the algorithm finds the burst j^* with the largest remaining effective load at time m. Then, it reserves all of the c_{j^*} upcoming MAC-layer slots in the channel for burst j^*.

The pseudo-code of PAL is shown in Fig. 3.2. In this pseudo-code, we assume that the following parameters are globally available: the duration of the scheduling window T_{sch}, the duration of a MAC-layer slot τ_{MAC}, the capacity vector $\tilde{C}$ (whose jth entry

```
    PAL:

1   schedule PAL {
2     burstSeq $\langle \mathscr{J}_{active} \rangle = \langle \phi \rangle$;
3     schedule $\mathbf{S} = \text{zeros}(N, T_{sch}/\tau_{MAC})$;
4     $m = 0$;
5     while($m < T_{sch}/\tau_{MAC}$) {
6       [$\langle \mathscr{J}_{active} \rangle$, nReservedSlots] = ComputeSchedule(&$\mathbf{S}$, $\langle \mathscr{J}_{active} \rangle, \tilde{C}, m$);
7       $m = m +$ nReservedSlots $+1$;
8     }
9     return $\mathbf{S}$;
10  }
```

Figure 3.2. Pseudo-code of the PAL heuristic

is equal to $R_j \tau_{MAC}$), and the entire set of bursts $\mathscr{J}$. Each element of $\mathscr{J}$ is a burst, which is a structure that contains the following fields: (1) burstID, which is the unique identification (ID) of a burst, (2) deviceID, which is the unique ID of the device that generated the burst, (3) numberOfBits (which is equal to a_j), (4) generationTime, which is the global index of the slot in which the burst is generated, and (5) deadline, which is the global index of the slot by the beginning of which the burst must be delivered.

On Line 2 in Fig. 3.2, the sequence of active bursts, denoted by $\langle \mathscr{J}_{active} \rangle$, is set to the null sequence. On Line 3, the schedule matrix $\mathbf{S}$ is set to the zero schedule matrix, whose number of rows equals N and whose number of columns equals the number of MAC-layer slots within the scheduling window, computed as T_{sch}/τ_{MAC}. The entry (i,m) of $\mathbf{S}$ equals 1 if slot m has been allocated to device $i \in \{1,\ldots,N\}$ and equals 0 otherwise. On Line 4, the global slot index m is initialized to 0. In each iteration m of the outer loop (Line 5) of PAL, the algorithm calls the ComputeSchedule function (Line 6), which computes a schedule given $\langle \mathscr{J}_{active} \rangle$ for the *current* slot m and returns $\langle \mathscr{J}_{active} \rangle$. On Line 6, the first input to the ComputeSchedule function is a pointer to the schedule matrix $\mathbf{S}$; the second input is $\langle \mathscr{J}_{active} \rangle$ for the current slot m; the third input is the capacity vector $\tilde{C}$; and the fourth input is the global slot index m. On Line 7, the global slot index is increased by nReservedSlots $+ 1$ in order to continue to schedule bursts at the next available MAC-layer slot. Finally, on Line 9, PAL returns the schedule matrix $\mathbf{S}$.

We now describe the ComputeSchedule function, which is shown in Fig. 3.3. On Line 2, the FindBurstsGeneratedInSlot function finds the sequence of bursts $\langle \tilde{\mathscr{J}} \rangle$, each of whose generation time falls in the current time slot $\tilde{m}$. (Recall that $\mathscr{J}$, which appears as an argument of the FindBurstsGeneratedInSlot function, denotes the *entire* set of bursts

```
1   [**burstSeq**, **int**] ComputeSchedule(schedule* $s$, burstSeq $\langle \mathscr{J}_{active} \rangle$,
                                                        vector $\tilde{C}$, int $\tilde{m}$) {
2     $\langle \tilde{\mathscr{J}} \rangle = \text{FindBurstsGeneratedInSlot}(\mathscr{J}, \tilde{m})$;
3     $\langle \mathscr{J}' \rangle = \text{Concat}(\langle \mathscr{J}_{active} \rangle, \langle \tilde{\mathscr{J}} \rangle)$;
4     $\langle \gamma[\tilde{m}] \rangle = \text{CalculateAverageLoad}(\langle \mathscr{J}' \rangle, \tilde{m})$;
5     $\langle \mathscr{J}_{sorted} \rangle = \text{Sort}(\langle \mathscr{J}' \rangle, \langle \gamma[\tilde{m}] \rangle)$;
6     nBursts = length($\langle \mathscr{J}_{sorted} \rangle$);
7     **for**($j = 0$; $j <$ nBursts; $j$++) {
8       $J_{\text{current}} = \langle \mathscr{J}_{sorted} \rangle[0]$;
9       $j = J_{\text{current}}$.burstID;
10      $i = J_{\text{current}}$.deviceID;
11      $a_j = J_{\text{current}}$.numberOfBits;
12      $d_j = J_{\text{current}}$.deadline;
13      $p_j = \lceil a_j / \tilde{C}[j] \rceil$;
14      **if**($(d_j < \tilde{m}) || (\tilde{C}[j](d_j - \tilde{m} + 1) < a_j)$) RemoveBurst($\langle \mathscr{J}_{sorted} \rangle, 0$);
15      **else** {
16        $(*s)[i, \tilde{m} : \tilde{m} + p_j - 1] = 1$;
17        RemoveBurst ($\langle \mathscr{J}_{sorted} \rangle, 0$);
18        return [$\langle \mathscr{J}_{sorted} \rangle, p_j$];
19      }
20    }
21    return [$\langle \mathscr{J}_{sorted} \rangle, 0$];
22  }
```

Figure 3.3. Pseudo-code of the ComputeSchedule function

over the current scheduling window.) On Line 3, $\langle \mathscr{J}' \rangle$ is formed by concatenating $\langle \mathscr{J}_{active} \rangle$ and $\langle \tilde{\mathscr{J}} \rangle$. On Line 4, the CalculateAverageLoad function calculates, for each burst j in $\langle \mathscr{J}' \rangle$, the average load at the beginning of slot $\tilde{m}$ as $\gamma_j[\tilde{m}] = a_j/(d_j - \tilde{m} + 1)$. On Line 5, the Sort function sorts $\langle \mathscr{J}' \rangle$ in descending order with respect to $\langle \gamma[\tilde{m}] \rangle$. On Line 6, the total number of bursts, denoted by nBursts, is calculated as the length of $\langle \mathscr{J}_{sorted} \rangle$. The *for* loop that begins on Line 7 attempts to schedule the first burst in $\langle \mathscr{J}_{sorted} \rangle$, where the burst can be scheduled within the scheduling window. We shall see that if the scheduling of the 0th element of $\langle \mathscr{J}_{sorted} \rangle$ has been attempted (ending in success or failure), $\langle \mathscr{J}_{sorted} \rangle$ will be updated by removing this 0th element. On Line 8, the current burst J_{current} is set to the 0th element of $\langle \mathscr{J}_{sorted} \rangle$. From this point onwards, we shall refer to the burst that is being scheduled as the "current burst". On Lines 9-12, the variables j, i, a_j, and d_j are set to the corresponding fields of the current burst. On Line 13, the processing time of the current burst p_j is calculated.

On Line 14, if the deadline of the current burst has expired or the total capacity between $\tilde{m}$ to d_j is not sufficient to process the current burst, the RemoveBurst function removes the current burst $\langle \mathscr{J}_{sorted} \rangle[0]$ from the sequence $\langle \mathscr{J}_{sorted} \rangle$. Otherwise, on Line 16, the slots between $\tilde{m}$ and $\tilde{m} + p_j - 1$ are allocated for device i; on Line 17, the RemoveBurst function removes the current burst from the sequence $\langle \mathscr{J}_{sorted} \rangle$; and on Line 18, the ComputeSchedule function returns the updated sequence $\langle \mathscr{J}_{sorted} \rangle$ and p_j as the number of reserved slots. On Line 21, if the ComputeSchedule function does not schedule any burst at slot $\tilde{m}$, it returns 0 as nReservedSlots and the updated sequence $\langle \mathscr{J}_{sorted} \rangle$.

3.3 Results

In this section, we present our results on the uplink cross-layer throughput as well as the computation time of the ARIMA, MLP, and LSTM forecasting models. We implemented the ARIMA and the MLP models in MATLAB on a 4.00 GHz DL380 Gen10 Intel Xenon-Gold 6138 48-Core Processor with 128 GB of RAM. In addition, we implemented the LSTM model in Python on the Google Colab platform using the TPU accelerator.

3.3.1 IoT Dataset

We divided the IoT devices into four distinct classes based on their traffic generation patterns. The four classes are: FBP, VBP, FBA and VBA. If the device generates a constant number of bits at all generation time instances, the traffic generation pattern is of type “Fixed Bit”; otherwise, it is of type “Variable Bit”. Furthermore, if the IoT device generates data at regular intervals, the traffic generation pattern is of type “Periodic”; otherwise, it is of type “Aperiodic”.

We obtained the first part of our data from the Temperature and Relative Humidity (DHT22), Light Dependent Resistor (LDR) and Air Quality (MQ135) sensors in our laboratory, measurements of which we collected over 2.5 months. We downloaded the second part of our data from the online data repository (University of California Irvine Machine Learning Repository, 2008)(De Vito, Massera, Piga, Martinotto, & Di Francia, 2008), which contains Air Quality measurements that have been collected from sensors over 1 year.

Table 3.1. Computed parameter and Φ values of the forecasting schemes for $N = 1000$ devices

	VBP		FBA		VBA	
	Parameters	Φ	Parameters	Φ	Parameters	Φ
ARIMA (p,d)	(2, 0)	0.0925	(1, 1)	0.1846	(3, 0)	0.0703
MLP $(n_1,n_2,\ldots n_E)$	(5, 1)	0.0763	(1)	0.2057	(1)	0
LSTM (h_{lstm},h_1,h_2)	(128, 64, 32)	0.0277	(128, 128, 32)	0.2290	(16, 128, 32)	0

3.3.2 Simulation Set-up

In our simulations, we set the values of the parameters of our JFS system as follows: $\tau_{MAC} = 0.1$ second; $T_{sch} = 1800$ seconds. The delay constraints (in seconds) of the applications that utilize the sensors are as follows: $\Delta_1 = 0.5$, $\Delta_2 = 1$, $\Delta_3 = 2$, $\Delta_4 = 180$, $\Delta_5 = 600$, $\Delta_6 = 3600$ with corresponding traffic generation periods (in seconds) $T_1 = T_2 = T_4 = 180$, $T_3 = T_5 = T_6 = 3600$. In addition, the data rate (in kbits/s) for the IoT devices are as follows: $R_1 = 33.33$, $R_2 = 40$, $R_3 = 33.33$, $R_4 = 50$, $R_5 = 50$ and $R_6 = 426.67$.

3.3.3 Parameter Tuning and Training

In this section, we present the structure of each forecasting model. For each model and each device class, we apply the following sequence of operations in order to tune the values of the parameters of the model: (1) We train the model on the traffic generation pattern of a randomly selected device. (2) We test the model on the traffic generation pattern of all of the devices in that device class. (3) We schedule the bursts of those devices within the scheduling window by using our PAL heuristic (Section 3.2.2).

We display the resulting values of the architectural parameters of each model in Table 3.1. In each of the subsections below, we discuss how we tune the parameters of each of the forecasting models.

ARIMA:

We implemented the ARIMA model using the Econometrics Toolbox of MATLAB, wherein we set the training function of the ARIMA model to maximum likelihood estimation (MLE). Recall that each feature index is the time index of the past sample relative to the current time. We set *MALags* equal to the sequence of feature indices that we determine by analyzing the empirical autocorrelation function of the traffic generation pattern. The resulting feature indices in our analysis were: 1, 2, 3 and multiples of 12 up to 120 for the VBP class; 1 and multiples of 23 up to 92 for the FBA class; and 1, 2, 3 for the VBA class. After we fixed the value of *MALags* for each device class, we set the intervals for exhaustive search as $[z_a^p, z_b^p] = [1,3]$ and $[z_a^d, z_b^d] = [0,3]$.

The resulting optimal values of p and d are given in the first row of Table 3.1. In the same row, in each column, we display the value of Φ that corresponds to the optimal (p,d) for a specific device class. We see that the largest value of Φ occurs for the FBA class.

MLP:

We set the activation function of the neurons in the hidden layers to the tangent hyperbolic (*tanh*) function. In addition, we did not use an activation function at the output layer. We set the initial value of each connection weight and each bias term to a random number distributed uniformly on the interval $[0,1]$.

We used the LM algorithm for training the MLP in order to minimize the MSE. We set the initial learning rate of LM to 0.001, the decrease factor of the learning rate to 0.1, the increase factor to 10, and the maximum learning rate to 10^{10}.

We set the number of neurons at the output layer equal to K_i for device i. Then, we found the local optimal architecture for the hidden layers via the MLP-SSA-Φ algorithm (Section 3.2.1). We set the parameter $\tilde{E}$ of this algorithm to 3. In the second row of Table 3.1, we display the resulting architecture of the MLP for each device class. In this table, each cell is of the form $(n_1, n_2, \ldots, n_E)$. We note that the local optimal hidden layer architecture is (1) for the "Aperiodic" device class. The reason is that as the traffic generation pattern becomes more unpredictable, MLP performs better by virtue of its simpler architecture.

LSTM:

In order to train the LSTM model, we used the Adam and MSE as the metric to be minimized. We implemented the same early stopping technique as for MLP.

We set the search intervals for the parameters h_{lstm}, h_1 and h_2 as follows: $[z_a^{lstm}, z_b^{lstm}] = [8, 256]$; $[z_a^{h_1}, z_b^{h_1}] = [32, 128]$ and $[z_a^{h_2}, z_b^{h_2}] = [16, 64]$. In each interval, we take only the values that are powers of 2. We display the resulting architecture for each device class under the LSTM model in the third row of Table 3.1. Each cell in this row is of the form (h_{lstm}, h_1, h_2). For LSTM, multiple architectures produce the same value of Φ. In Table 3.1, we report the simplest LSTM architecture that achieves the lowest value of Φ.

3.3.4 Comparison of the Throughput Performance of Forecasting Models

In this section, we compare the performance of forecasting models with respect to the uplink cross-layer throughput (Section 1.4) of JFS. Furthermore, we display the throughput under perfect forecasts, which provides an upper bound on the performance of our forecasting schemes.

In order to understand the relationship between the IoT devices classes and throughput, we first set up a simulation in which the network consists of IoT devices from a single device class. We note that no forecasting is needed for the FBP class because devices in this class generate a constant number of bits at regular intervals. Hence, once the initial shift in timing has been communicated to the Gateway, the entire traffic generation pattern of that device is known. Therefore, in our simulations, we shall examine the performance of forecasting schemes only for the VBP, FBA and the VBA classes.

First, we set up a simulation in which all of the IoT devices fall in the VBP class. The throughput for this scenario is displayed in Fig. 3.4. In this figure, we see that LSTM outperforms the other three models with respect to network throughput with a gap of approximately 0.1. The reason for the superior performance of LSTM is that traffic in the VBP class has seasonal trends, which LSTM captures effectively. ARIMA and MLP are not able to capture the seasonal trends as well as LSTM does because the LSTM model has forget and update gates in its structure which ARIMA and MLP lack. Note that the throughput when LSTM is used as the forecasting scheme is the same as that under perfect forecasting up to 700 devices. Beyond 700 devices, LSTM slightly

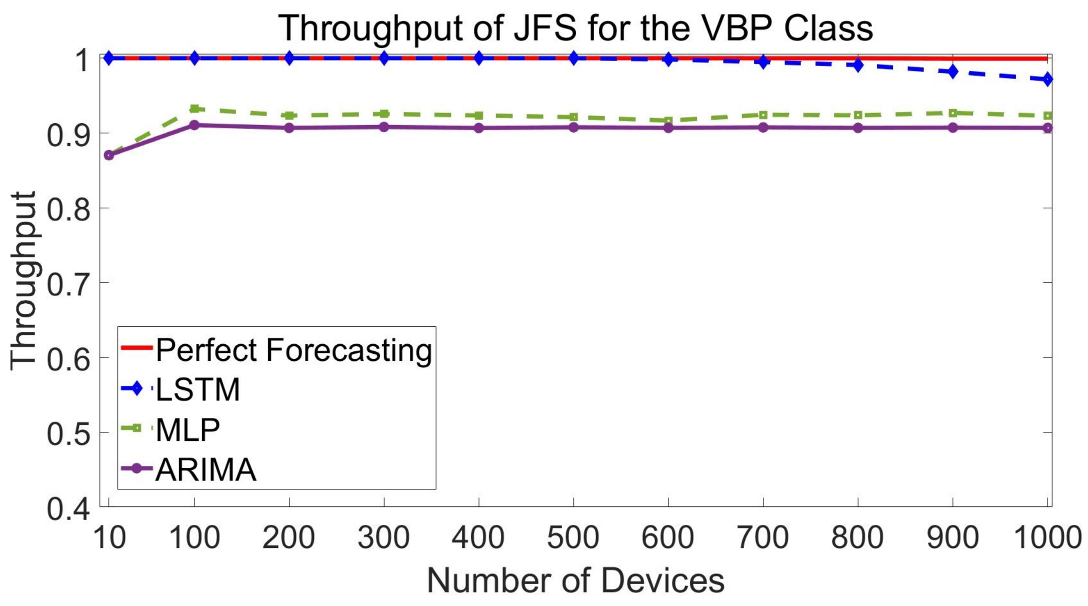

Figure 3.4. Throughput of JFS for the VBP class

underperforms when compared with perfect forecasting; however, the gap is relatively small.

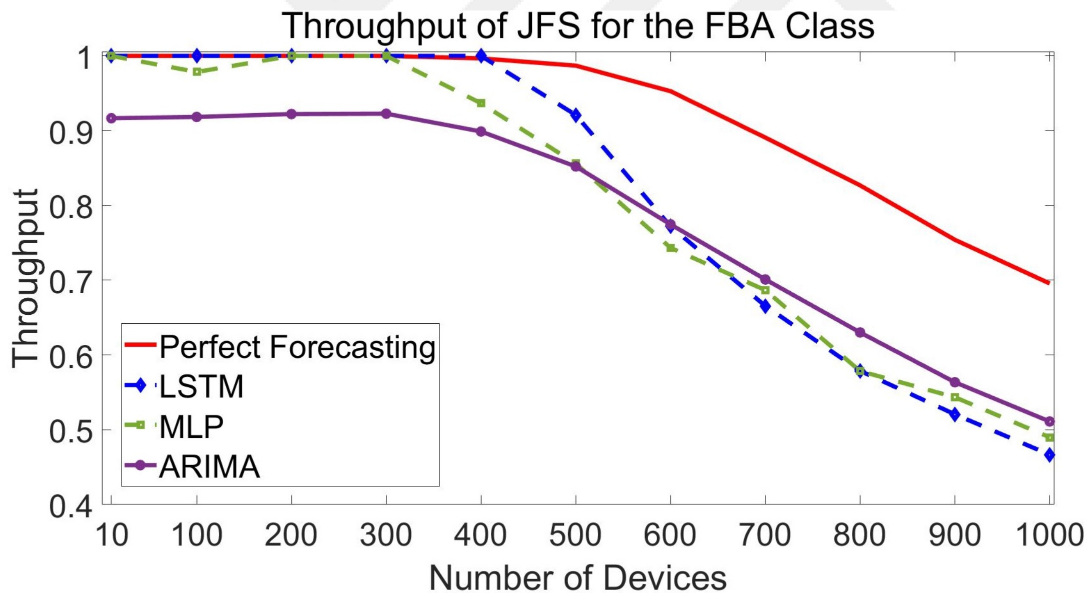

Figure 3.5. Throughput of JFS for the FBA class

Second, we set up a simulation in which all of the IoT devices fall in the FBA class. Fig. 3.5 displays the throughput of the JFS system for this device class under each forecasting model. We see that the gap between the throughput under perfect forecasting and that under the other models increases beyond 400 devices. We examined the resulting schedule of each forecasting scheme and found that typically LSTM overallocates and ARIMA underallocates the number of MAC-layer slots that are dedicated to each burst. The impact of the overallocation of LSTM on throughput is that in the heavily loaded regime beyond 400 devices, the gap between the performance of LSTM and that under perfect forecasting widens dramatically. The reason is that in the heavily loaded regime, overallocation of the number of slots for each burst prevents adjacent bursts from being scheduled. In contrast, the performance gap between ARIMA and perfect forecasting remains roughly constant except between 300 and 500 devices, where it widens only slightly. Thus, ARIMA displays a robust performance as the traffic load increases because it does not suffer from the effects of overallocation.

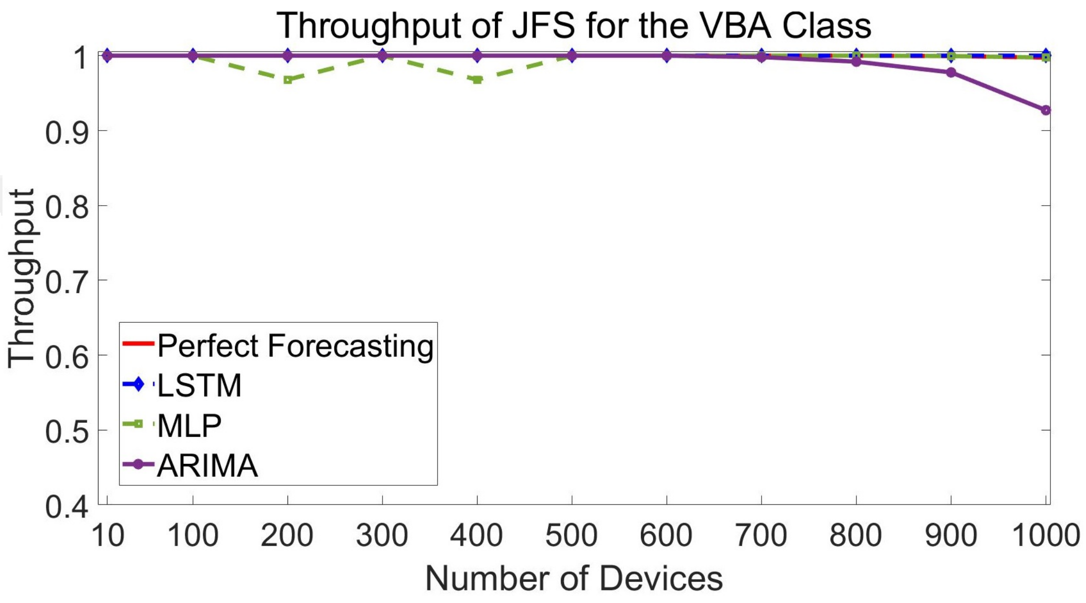

Figure 3.6. Throughput of JFS for the VBA class

Third, we set up a simulation in which all of the IoT devices fall in the VBA class. Fig. 3.6 displays the throughput for this device class under each forecasting model. We note that, compared with Fig. 3.4 and Fig. 3.5, the throughput for the VBA class remains roughly constant near 1 for almost all of the forecasting models up to 700 devices. The reason is that for the VBA class, the distribution of bursts over time is more sparse than those for the VBP and the FBA classes; hence, the network remains in the lightly loaded regime. Furthermore, beyond 700 devices, the throughput for the ARIMA model decreases beyond that achieved under perfect forecasts. The reason is that ARIMA overallocates the number of MAC-layer slots for VBA traffic.

Finally, we set up a system simulation that has an equal proportion of IoT devices from each of the FBP, VBP, FBA and VBA classes. In Fig. 3.7, we display the throughput up to 1000 devices. We see that all of our forecasting schemes perform close to perfect

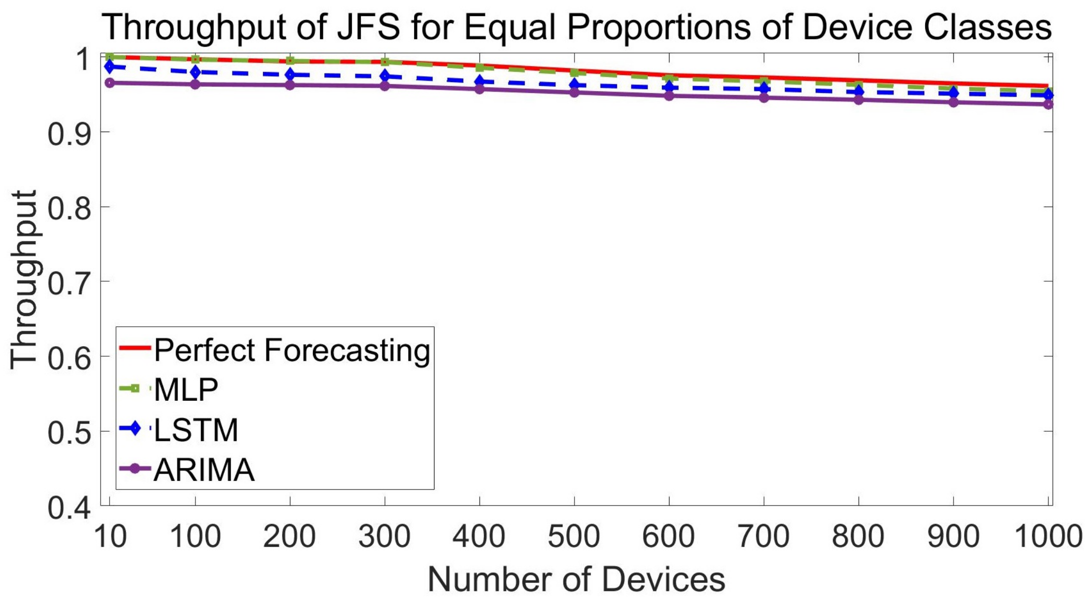

Figure 3.7. Throughput of JFS in a simulation environment with an equal proportion of devices from the FBP, VBP, FBA and VBA classes

forecasting. We note that the impact of the performance of JFS for the FBA class on this simulation is delimited by the fact that only 25% of the total number of devices belong to the FBA class in this simulation. Up to 250 devices, MLP and LSTM perform very close to the perfect forecasting bound in Fig. 3.5. The inferior performance of ARIMA in Fig. 3.5 translates into a performance loss in Fig. 3.7 that is proportional to the number of devices in the FBA class in this simulation.

3.3.5 Comparison of Forecasting Models with Respect to Training and Execution Times

In this section, we present our results on the training and execution times of our forecasting models. We report the mean and the standard deviation (STD) for each device class over all of the devices in 10 simulation runs.

Training Time:

We display the training time per device of each forecasting model for each device class in Table 3.2. We see that there is a significant difference between the training time of LSTM and those of the other models. The reason is two-fold: (1) The number of parameters for LSTM is greater than those of the other models, and (2) the architecture of LSTM is more complex than those of the other models (Table 3.1).

Table 3.2. Training time of forecasting models per IoT device

	VBP		FBA		VBA	
	Mean	STD	Mean	STD	Mean	STD
ARIMA	20.986	0.198	4.014	0.2133	0.299	0.015
MLP	2.826	0.919	0.168	0.057	1.162	0.882
LSTM	591.856	6.599	44.659	15.584	11.891	3.727

Execution Time:

In Table 3.3, we show the mean and the standard deviation of the execution time per device. We see that the mean of the execution time of all of the forecasting models are less than 1 second. Furthermore, the execution time of the MLP model is two orders of magnitude lower than those of ARIMA and LSTM for the VBP and FBA classes. For the VBA class, the execution time of MLP is approximately 1/5 that of ARIMA and two orders of magnitude lower than that of LSTM.

Table 3.3. Execution time of forecasting per IoT device

	VBP		FBA		VBA	
	Mean	STD	Mean	STD	Mean	STD
ARIMA	0.447	0.012	0.088	0.006	0.004	0.0008
MLP	0.004	0.001	0.0009	0.0003	0.0008	0.0003
LSTM	0.607	0.024	0.088	0.005	0.087	0.009

3.3.6 Computation Time of the PAL Scheduling Heuristic

The execution time of JFS depends not only on the execution time of the bank of forecasters (all of which run in parallel) but also on the execution time of the scheduler. In order to understand whether our JFS system can be implemented in practice, we now undertake an examination of the computation time of PAL, which is our scheduling heuristic.

In Fig. 3.8, we see that the computation time of PAL grows approximately linearly with the total number of devices. For 1000 devices, the computation time is roughly 0.1132 seconds with a standard deviation of 0.0196 seconds. (Each error bar shown in Fig. 3.8 corresponds to one standard deviation in each vertical direction.)

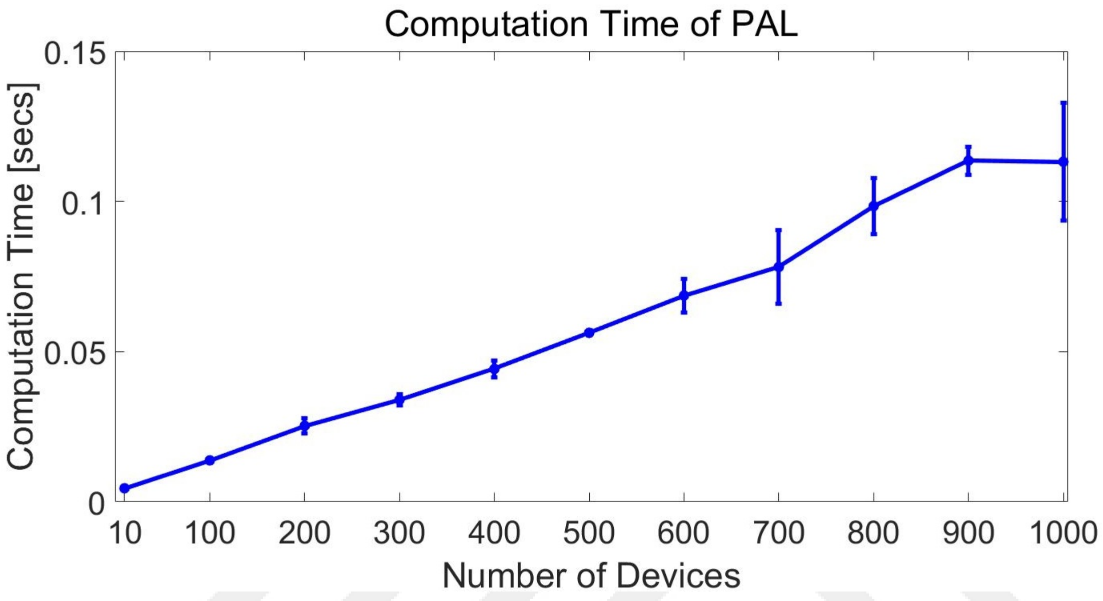

Figure 3.8. Computation time of PAL under perfect forecasting

We denote the execution time of forecaster i in Fig. 3.1 by T_i^{fc}. Then, the execution time of the bank of parallel forecasters, denoted by T^{bank}, is the maximum execution time over all of the forecasters in Fig. 3.1; that is, $T^{bank} = \max_{i \in \mathcal{N}} T_i^{fc}$. Note that scheduling cannot start before the latest forecaster finishes its task. Joint forecasting-scheduling must finish by the earliest generation time over all of the bursts in the upcoming scheduling window. The earliest generation time over all of the bursts in that window is within $T_{min} \equiv \min_i T_i$ seconds of the current time. We denote the sum of T^{bank} and the computation time of the scheduler by T_{JFS}. Thus, for the JFS system to work in real time, T_{JFS} must be smaller than T_{min}.

We form a conservative estimate of T_i^{fc} as the sum of the mean and two standard deviations for T_i^{fc}, using the data in Table 3.3. Similarly, we estimate the execution time of PAL as the sum of the mean and two standard deviations in Fig. 3.8. Then, for a total of 1000 devices, an upper bound on T_{JFS} is estimated to be 0.1584 seconds for MLP, which is the fastest forecaster. Note that only 4% of this upper bound is spent on forecasting and 96% on scheduling.

3.4 Summary

In this chapter, we showed that among all of the forecasting schemes, MLP provides a reasonable trade-off between uplink cross-layer throughput and execution time. When all of the FBP, VBP, FBA and VBA classes are represented in equal proportions in a network, the throughput performance loss of MLP with respect to perfect forecasts is minimal. However, MLP underperforms significantly in network throughput when all of the devices in the network fall in the FBA class. In this case, a combination of LSTM and ARIMA outperforms MLP. Hence, our key conclusion is that the choice of the forecasting scheme depends heavily on the proportions with which distinct device classes are represented in the network. In a network where these proportions change dynamically, the selection of the forecasting scheme must be dynamic as well.

Chapter 4

A MULTI-SCALE ALGORITHM FOR JOINT FORECASTING-SCHEDULING

4.1 Introduction

In this chapter[1], we develop a novel algorithm, called the Multi-Scale Algorithm (MSA) for the JFS system. Our goal in designing this algorithm is to solve the Massive Access Problem while satisfying the wide range of delay constraints of IoT applications.

In Chapter 3, we showed that the JFS system under perfect forecasts achieves a cross-layer network throughput of 1 for at least up to 500 devices, which are connected to the Gateway *G*. However, in that chapter, we also saw that the performance of the JFS system decreases due to the forecasting errors as the number of devices increases.

The main contribution of this chapter is the development of a novel algorithm that operates at multiple time scales to perform joint forecasting and scheduling of IoT devices at the MAC layer. The main advantage of such a methodology is three-fold: (1) The amount of control signaling overhead can be kept to a minimum. (2) The delays that occur in traditional protocols due to contention, collisions and handshaking can be avoided. (3) If the forecasts are accurate, high network performance can be achieved via reservation-based scheduling.

In this chapter, we show that an MLP-based architecture for forecasting, combined with exact scheduling solutions, achieves high network throughput while satisfying the diverse delay deadlines of the IoT devices. Furthermore, such performance can be obtained at practically feasible time and space computational complexity such that it can be implemented at an IoT Gateway.[2] We compare the throughput and energy consumption of MSA with those of three protocols: (1) RAB, which is a reactive,

[1]The technical content in this chapter has been published as a journal paper (Rodoplu, Nakıp, Eliiyi, & Güzelis, 2020) in the IEEE Internet of Things Journal.

[2]We envision that this IoT Gateway is either co-located with the base station or acts as a relay to the base station. In the framework of this chapter, the IoT devices are connected not to the base station but rather to this IoT Gateway and do *not* implement the LTE standard.

joint access-scheduling protocol that we have designed that combines features of ACB (Ghavimi & Chen, 2015) and reservation-based scheduling (Yu et al., 2013). (2) PAL, which is a proactive protocol that we have designed that utilizes non-preemptive priority scheduling(Stallings, 2009) such that the priority across devices is determined based on the time-averaged forecast remaining traffic load until the delay deadline. (3) E-PRV-BO protocol, which is a proactive protocol that enhances the PRV-PO protocol in (Hammad et al., 2017). We show that MSA significantly outperforms all of these three protocols beyond 3000 devices. Furthermore, we show that the percentage control overhead of MSA remains less than 1.5% for up to 6650 devices. Finally, we present a comparison of the performance of MSA against those of the above three protocols under the IEEE 802.11ah standard.

The rest of this chapter is organized as follows: In Section 4.2, we describe our system design. In Section 4.3, we present our multi-scale algorithm for joint forecasting-scheduling. In Section 4.4, we present our results on the network performance of this algorithm. In Section 4.5, we discuss how our framework can be extended to scenarios in which IoT devices join or leave the network. In Section 4.6, we present our conclusions.

4.2 System Design

In our system, the Gateway G forecasts the traffic generation pattern $\{X_i[k]\}$ of each device i over a scheduling window of duration T_{sch}. It then aims to schedule the devices' uplink transmissions free of any collisions over this window based on its forecasts. We assume a Frequency Division Duplexing (FDD) system. On a downlink broadcast channel, G communicates to each device i the exact set of slots, denoted by $\mathscr{A}_i$, that are allocated to i on the uplink. If i is always awake, this information may be communicated to i just before its uplink transmission is to occur; however, if i sleeps in order to conserve energy, $\mathscr{A}_i$ must be communicated to i by G in advance. In our system, on the downlink, G tells each i its $\mathscr{A}_i$ for i's *next forecast burst*.

Whenever each device i transmits to G, it piggybacks a compressed representation of its actual generated traffic pattern since its last transmission. Based on this information, G records for each i, that device's actual traffic generation pattern $\{X_i[k]\}$, which G uses for forecasting the *future* traffic pattern $\{\hat{X}_i[k]\}$ of that device. (We assume that strong forward error correction for short packets (Durisi, Koch, & Popovski, 2016) is utilized at the physical layer such that we can assume that an uplink transmission that is scheduled free of collisions is received successfully.)

4.3 Joint Forecasting-Scheduling

We now present a novel joint forecasting-scheduling algorithm, in which G runs a forecaster for each device i, based on which it schedules the uplink traffic of all of the devices *in advance* over the scheduling window of duration T_{sch}.

4.3.1 Multi-Scale Forecasting

For each device i, we define the "traffic generation period" of i, denoted by T_i, as the minimum duration between successive traffic generation instances of i such that all traffic generation times are multiples of this duration. (This assumption is justified for all IoT devices that sample at regular intervals.) We define (in absolute time units, such as seconds) the "accurate forecasting window" T_i^{af} of device i as the time interval, starting at the current time, over which G can form accurate forecasts of i's traffic. For each i's traffic, accurate forecasts are possible typically up to a fixed number of samples, denoted by K_i; thus, $T_i^{af} = K_i T_i$ for every i. Now, T_{sch}, which is common for all devices, must satisfy $T_{sch} \leq \min_{i \in \mathcal{N}} T_i^{af}$ since an optimized schedule is possible only when all of the devices' forecasts across the scheduling window are accurate. Thus, the longest possible T_{sch} equals $T_{min}^{af} \equiv \min_i T_i^{af}$. However, when T_{sch} is set to T_{min}^{af}, an optimized schedule for all devices that spans their respective $\{T_i^{af}\}$ cannot be computed since the traffic pattern of any device i for which T_i^{af} is longer than T_{min}^{af} cannot be represented over this T_{sch}.

A key novelty in this chapter is that we solve the above problem by representing $\{X_i[k]\}$ for each i at *multiple time scales* by summing the $\{X_i[k]\}$ over increasingly larger, non-intersecting "blocks" (of time slots) recursively as follows: At the bottom level (or scale), which we shall call Level 0, the block size (or duration) is equal to τ_{MAC}, which denotes the duration of a MAC-layer slot. The amount of data that falls in Block b at this level is given by $X_i[b] = X_i[k]$, where each block b is the MAC-layer slot k. Now, given a set of $V_l M_l$ blocks at level l, we group these blocks in V_l non-overlapping, contiguous sets of M_l blocks in each set. We say that each of the blocks at level l that are grouped to form block B at level $l+1$ is a "subblock" of B. The set of subblocks of B is denoted by S_B. Furthermore, let $X_i(B)$ and $X_i(b)$ denote the number of bits of device i in block B (at level $l+1$) and in any of its subblocks b (at level l), respectively.

Then,

$$X_i(B) = \sum_{b \in S_B} X_i(b)$$

We terminate the recursion at level L.[3] (Hence, the total number of levels, including Level 0, is $L+1$.) The Gateway G forecasts the traffic of device i at all levels $l \geq 0$. We let $\hat{X}_i(b)$ denote the forecast of the number of bits of device i in block b at level l. We define the "accurate forecasting window" of device i at level l, starting at the current time $t = 0$, as $[0, \tilde{T}_{il}^{af}]$, where $\tilde{T}_{il}^{af}$ is the length of the absolute time interval (measured in seconds) over which G can form accurate forecasts of device i's traffic at level l. A key finding of the empirical part of this chapter (Section 4.4) is that when the parameters of our algorithm are determined as will be described in Section 4.3.2, $\tilde{T}_{il}^{af}$ is a monotonically increasing function of l for each i. This allows us to represent the traffic generation patterns over scheduling windows whose lengths (in seconds) grow as traffic is represented at increasingly coarser levels. This property will be exploited in the next section.

4.3.2 Multi-Scale Forecasting-Scheduling Algorithm

Algorithm Parameters:

For our multi-scale joint forecasting-scheduling algorithm (which we call Multi-Scale Algorithm or "MSA" for short), the total number of levels $L+1$, the block size B_l (in seconds) at each level l and the length of the scheduling window (in seconds) $T_{sch}^{(l)}$ at that level are determined as follows:

First, the latency constraints $\{\Delta_j\}$ are sorted in non-decreasing order. Without loss of generality, order these as $\Delta_1 \leq \Delta_2 \leq \cdots \leq \Delta_{\tilde{R}}$, assuming that there are $\tilde{R}$ distinct values for latency. (In ordering, ties between equal latency values are broken arbitrarily.) Second, we uniformly quantize these values into equivalence classes (which we call "bins") as follows: The first bin, indexed as $q = 1$, has a latency constraint $\tilde{\Delta}_1 = \Delta_1$. Next, using a quantization interval of length $T_{min} \equiv \min_{i \in \mathcal{N}} T_i$, for integers $q \geq 1$, latency intervals of the form $[\tilde{\Delta}_1 + (q-1)T_{min}, \tilde{\Delta}_1 + qT_{min}]$ partition the $\{\Delta_j\}$ into bins. If no Δ_j falls in a bin, that bin is eliminated. The remaining bins are numbered in increasing order from 1 to Q. We associate a latency constraint $\tilde{\Delta}_q = \min\{\Delta_j | \Delta_j \in \text{Bin } q\}$ with bin

[3]The procedure for the determination of L is specified in Section 4.3.2.

q. Furthermore, we let $\tilde{\Delta}_{min} \equiv \tilde{\Delta}_1$ and $\tilde{\Delta}_{max} \equiv \tilde{\Delta}_Q$.

Second, given $\tilde{\Delta}_1 < \tilde{\Delta}_2 < \cdots < \tilde{\Delta}_Q$, there are three possible cases: (Case 1) $T_{min} \geq \tilde{\Delta}_{max}$: In this case, we set $L = 0$ and $B_0 = \tau_{MAC}$. (Case 2) $\tilde{\Delta}_{min} \leq T_{min} < \tilde{\Delta}_{max}$: In this case, let F denote the number of $\tilde{\Delta}_q$'s that are less than or equal to T_{min}. Then, we set $L = Q - F$. Furthermore, we set $B_0 = \tau_{MAC}$, and $\forall l \geq 1$, $B_l = \tilde{\Delta}_{F+l}$. (Case 3) $T_{min} < \tilde{\Delta}_{min}$: In this case, we set $L = Q$. Furthermore, we set $B_0 = \tau_{MAC}$, and $\forall l \geq 1$, $B_l = \tilde{\Delta}_l$.

Third, we set $T_{sch}^{(l)} = \min_{i \in \mathscr{N}} \{\tilde{T}_{il}^{af}\}$ for each level l.

Scheduling and Load Balancing:

The MSA has two general subroutines: Scheduling, which is used at Level 0, and Load Balancing over Blocks (LBB), which is used at levels 1 to L.[4]

First, for scheduling, in this chapter, we shall use a particular technique, called "pre-emptive scheduling of a single machine to minimize the weighted number of late jobs" (Potts & Van Wassenhove, 1988), which we call Job Scheduling with Time Windows (JSTW), for short. We apply JSTW to our case by modeling each burst (as defined in Section 1.4) as a "job", which produces the optimization program below.

JSTW:

$$\max \textstyle\sum_{j \in \mathscr{J}} a_j u_j$$

subject to:

1. $\sum_{j \in \mathscr{J}} y_{jk} \leq 1 \qquad \forall k \in \mathscr{K}_+$

2. $\sum_{k=r_j}^{d_j} y_{jk} = u_j p_j \qquad \forall j \in \mathscr{J}$

At Level 0, JSTW is used to produce schedules of all devices over $[0, T_{sch}^{(0)}]$. Above, a_j, p_j, r_j and d_j are parameters, and y_{jk} and u_j are binary decision variables. Furthermore, $\mathscr{J}$ is the set of bursts to be sent to G over $[0, T_{sch}^{(0)}]$, and a_j is the number of bits in burst j. The parameter r_j is the (MAC-layer) slot in which burst j is generated, and d_j is the slot by the end of which all of the bits of burst j have to be sent to G. For each j, $d_j \leq \lceil T_{sch}^{(0)} / \tau_{MAC} \rceil$. Each y_{jk} is defined only on the interval $r_j \leq k \leq d_j$, and $y_{jk} = 1$ if any of the bits of burst j are scheduled to be sent in slot k and $y_{jk} = 0$ otherwise. Above,

[4]We note that any scheduling algorithm may be used for the first subroutine, even though we make a particular choice for the purposes of this chapter.

$\mathscr{K}_+$ is the set of k's for which at least one y_{jk} has been defined. Furthermore, $u_j = 1$ if all of the bits in burst j are sent by the end of slot d_j, and $u_j = 0$ otherwise. Finally, $p_j \equiv \lceil a_j / C_{f(j)} \rceil$, is the "processing time" of burst j, where C_i denotes the maximum number of bits that i can send to G in a slot (when it is the only device scheduled for uplink transmission), and the function $f(j)$ maps burst j to the device that generated the burst. Thus, the objective is to maximize the total number of bits in *completed* bursts that are sent to G such that (1) in any given slot, at most a single burst can be scheduled[5] and (2) the total number of slots allocated to burst j equals p_j if all of the bits of j are sent by d_j, and no slot is allocated to j otherwise.

Second, at each level $l : 1 \leq l \leq L$, LBB is used to distribute the total number of forecast bits over the set of blocks at level l on $[0, T_{sch}^{(l)}]$. The optimization program appears below.

LBB:

$$\min H^{(l)}$$

subject to:

1. $\sum_{b=r_j^{(l)}}^{d_j^{(l)}} z_{jb} = 1 \qquad \forall j \in \mathscr{J}^{(l)}$

2. $\sum_{j=1}^{J^{(l)}} p_j^{(l)} z_{jb} \leq H^{(l)} \qquad \forall b \in \mathscr{B}_+^{(l)}$

Above, $p_j^{(l)}$, $r_j^{(l)}$, and $d_j^{(l)}$ are parameters, and $H^{(l)}$ and z_{jb} are decision variables. Whenever $\hat{X}_i(b) > 0$ for any device i in block b at level l, we say that device i generates a "job" j at level l with $\hat{X}_j^{(l)}$ bits equal to $\hat{X}_i(b)$. Above, $\mathscr{J}^{(l)}$ is the collection of all such jobs at level l over all devices $\mathscr{N}$. Furthermore, $r_j^{(l)}$ is the block at level l in which job j is generated, and $d_j^{(l)}$ is the block by the end of which the bits in that job must be successfully delivered to G. Thus, $d_j^{(l)} = r_j^{(l)} + \lceil \Delta_j / B_l \rceil - 1$, where Δ_j is the latency constraint of the bits in job j. Above, each z_{jb} is defined only on the interval $[r_j^{(l)}, d_j^{(l)}]$. Furthermore, $z_{jb} = 1$ if all of the bits in job j are assigned to block b (in which case we say that job j has been "loaded" onto block b) and $z_{jb} = 0$ otherwise. Above, $p_j^{(l)} \equiv \lceil \hat{X}_j^{(l)} / C_{f(j)} \rceil$, and $\mathscr{B}_+^{(l)}$ is the set of blocks at level l for which at least one z_{jb} has been defined. Thus, the objective is to minimize the "height" $H^{(l)}$ at level l such

[5] This condition is implied by the facts that (1) bursts, by definition, are in close proximity in time, and (2) in order to avoid collisions, in our design, we do not choose to schedule more than 1 device in any given slot.

that (1) each job is loaded onto exactly one block such that its delay deadline is met, and (2) the total number of MAC-layer slots required to deliver the bits in all of the jobs loaded onto block b is at most the height $H^{(l)}$ for each block b. In summary, this algorithm balances the load across the blocks on $[0, T_{sch}^{(l)}]$.

Multi-Scale Algorithm:

Fig. 4.1 shows the pseudo-code for our MSA. The MSA is a *real-time*, recursive algorithm that begins at Level L and produces an uplink MAC-layer schedule at Level 0. On the scheduling window $[0, T_{sch}^{(L)}]$, this algorithm is called as MultiScaleAlgorithm($\mathscr{N}$, $\mathbf{B}^{(L)}$), where $\mathbf{B}^{(L)}$ is the single block on this window at Level L. In Fig. 4.1, on Line 2, $\mathbf{W}$ is the set of fixed forecasting parameters ("weights") for all i, obtained via offline training. (When MLP is used as the forecasting method as in Section 4.4.2, $\mathbf{W}$ is a 5-dimensional (5D) matrix whose entry (l, i, e, n, m) is the connection weight that emanates from neuron m at neural layer $e-1$ and enters neuron n at neural layer e for device i at level l of the MSA.) Furthermore, $\mathbf{X}$ is the 2D matrix of *past* $\{X_i[k]\}$ (where row i is the device and column k is the slot index), communicated to G (as in Section 4.2).

```
   Multi-Scale Algorithm:
1  void MultiScaleAlgorithm($D, \mathbf{B}$) {
2    $[\mathbf{w}^{(\mathbf{B})}, \mathbf{x}^{(\mathbf{B})}] =$
     LoadForecastingWeightsAndPastTraffic($D, \mathbf{W}, \mathbf{X}, \mathbf{B}$);
3    $\hat{\mathbf{x}}^{(\mathbf{B})} =$ ForecastFutureTraffic($D, \mathbf{w}^{(\mathbf{B})}, \mathbf{x}^{(\mathbf{B})}, \mathbf{B}$);
4    if($\mathbf{B}$.depth() $== L$)
5      $\mathbf{S}^{(\mathbf{B})} =$ ScheduleDevicesAtMACLayer($D, \hat{\mathbf{x}}^{(\mathbf{B})}$);
6    else {
7      $[\langle\tilde{D}\rangle, \langle\mathbf{b}\rangle] =$ BalanceLoad($D, \hat{\mathbf{x}}^{(\mathbf{B})}, \mathbf{B}$);
8      for($n = 0; n <$ length($\langle\tilde{D}\rangle$); $n$++) {
9        MultiScaleAlgorithm($\langle\tilde{D}\rangle[n], \langle\mathbf{b}\rangle[n]$);
10     }
11   }
12 }
```

Figure 4.1. Pseudo-code for the Multi-Scale Algorithm

We now describe a single recursive step of MSA: The index $\mathbf{B}$ locates the block globally within the scheduling window $[0, T_{sch}^{(L)}]$. (In addition, $\mathbf{B}$ is a structure that contains the sequence of its subblocks and the duration (in seconds) of any of its subblocks, which is identical across all of the subblocks of $\mathbf{B}$.) For a given vector of devices D, on Line

2, the LoadForecastingWeightsAndPastTraffic function, whose pseudo-code appears in Fig. 4.2, extracts those weights $\mathbf{w}^{(\mathbf{B})}$ in $\mathbf{W}$ and those past traffic patterns $\mathbf{x}^{(\mathbf{B})}$ in $\mathbf{X}$ that belong to all of the devices in D that fall in block $\mathbf{B}$. (The pseudo-code for each of the functions that the MultiScaleAlgorithm calls will be described in detail below.) On Line 3, the function ForecastFutureTraffic forecasts the future traffic $\hat{\mathbf{x}}^{(\mathbf{B})}$ of all of the devices in D that fall in block $\mathbf{B}$, based on the weights $\mathbf{w}^{(\mathbf{B})}$ and the past traffic patterns $\mathbf{x}^{(\mathbf{B})}$. (We emphasize that the designer chooses the forecasting scheme for the function ForecastFutureTraffic; the MSA does *not* assume any particular forecasting scheme. A particular implementation of the ForecastFutureTraffic function, based on MLP, appears in Fig. 4.3 and shall be detailed in Section 4.4.2.) If the entire $\mathbf{X}$ required to perform $\lceil T_{sch}^{(l)}/B_l \rceil$-step ahead prediction of $\hat{\mathbf{x}}^{(\mathbf{B})}$ has not yet been communicated to G (see Section 4.2), the function ForecastFutureTraffic stalls until this prediction becomes possible. On Lines 4-5, if MSA has arrived[6] at Level 0, the function ScheduleDevicesAtMACLayer schedules the devices in D at the MAC Layer. (A particular implementation of this function that employs JSTW is shown in Fig. 4.4. However, we note that any collision-free burst scheduling algorithm may be used in the place of JSTW.) The $N \times \lceil T_{sch}^{(0)}/\tau_{MAC} \rceil$ matrix $\mathbf{S}^{(\mathbf{B})}$ (on Line 5) is the "scheduling matrix" for block $\mathbf{B}$, whose entry (i,k) equals 1 if device i is granted MAC-layer slot k over the current scheduling window at Level 0, and equals 0 otherwise. If MSA has not arrived at Level 0, on Line 7, the function BalanceLoad balances the traffic load in $\hat{\mathbf{x}}^{(\mathbf{B})}$, which produces a sequence of vectors, denoted by $\langle \tilde{D} \rangle$, over the temporally ordered sequence of all of the subblocks $\langle \mathbf{b} \rangle$ of block $\mathbf{B}$. (The earliest (left-most) subblock appears as the first element and the latest (right-most) subblock appears as the last element of $\langle \mathbf{b} \rangle$.) The nth element of $\langle \tilde{D} \rangle$, which shall be denoted by $\langle \tilde{D} \rangle[n]$, is the vector of devices each of which falls in the nth subblock in the sequence $\langle \mathbf{b} \rangle$ as a result of load balancing. On Lines 8-9, MSA (recursively) calls itself on each of these vectors of devices, $\langle \tilde{D} \rangle[n]$. When the MSA in Fig. 4.1 terminates with a MAC-layer schedule on $[0, T_{sch}^{(L)}]$, it will be called again on the next scheduling window at Level L.

We now begin a description of each of the functions called by MSA. Fig. 4.2 displays the pseudo-code of the LoadForecastingWeightsAndPastTraffic function. On Line 2, the duration of the subblock $\mathbf{b}$ of $\mathbf{B}$, is extracted from the structure $\mathbf{B}$. On Line 3, the total number of subblocks *nPastSubblocks*, in the past traffic generation pattern $\mathbf{X}$, that have passed until the beginning of the current block $\mathbf{B}$ is calculated. On Line 4, the matrix of the traffic generation pattern for all of the devices in vector D is initialized to the zero matrix, whose number of rows is the length of the vector D, and whose number

[6]$\mathbf{B}$.depth() $= L - l$, where l is the level at which block $\mathbf{B}$ is located.

```
   **LoadForecastingWeightsAndPastTraffic:**
1  [4D-matrix, 2D-matrix]
   LoadForecastingWeightsAndPastTraffic($D, \mathbf{W}, \mathbf{X}, \mathbf{B}$) {
2    $|\mathbf{b}|$ = $\mathbf{B}$.GetSubblockDuration();
3    $nPastSubblocks = \text{size}(\mathbf{X}, 2)/(\lceil |\mathbf{b}|/\tau_{MAC} \rceil)$;
4    $\mathbf{x} = \text{zeros}(\text{length}(D), nPastSubblocks)$;
5    $\mathbf{w} = [\,]$;
6    for($i' = 0; i' < \text{length}(D); i'$++) {
7      $\tilde{\mathbf{X}} = \text{Reshape}(\mathbf{X}[D[i'], :], \lceil |\mathbf{b}|/\tau_{MAC} \rceil,$
                                              $nPastSubblocks)$;
8      $\mathbf{x}[i', :] = \text{Sum}(\tilde{\mathbf{X}}, 1)$;
9      $\mathbf{w} = \text{Concat}(\mathbf{w}, \mathbf{W}[L - \mathbf{B}.\text{depth}(), D[i'], :, :, :], 1)$;
10   }
11   return $[\mathbf{w}, \mathbf{x}]$;
12 }
```

Figure 4.2. Pseudo-code for the LoadForecastingWeightsAndPastTraffic function

of columns is the number of past subblocks. On Line 5, the 4D matrix of connection weights $\mathbf{w}$ for the current block $\mathbf{B}$ is initialized to the empty matrix. The entry (i', e, n, m) of $\mathbf{w}$ is the connection weight that emanates from neuron m at neural layer $e-1$ to neuron n at neural layer e for the device with local index i'. (In this notation, $e \geq 1$, and $e = 0$ corresponds to the input layer.) For each device in vector D, on Line 7, the matrix $\tilde{\mathbf{X}}$ is formed by reshaping (as the term is used in MATLAB) the past traffic generation pattern of the i'th device in D such that the number of rows of the reshaped matrix $\tilde{\mathbf{X}}$ equals the number of MAC-layer slots in a subblock of $\mathbf{B}$, and its number of columns equals the number of past subblocks. The entry in the k'th row and the bth column of $\tilde{\mathbf{X}}$ is the number of bits generated by device $D[i']$ in the k'th MAC-layer slot of the bth subblock of $\mathbf{B}$. On Line 8, the number of bits generated by device $D[i']$ in each of the past subblocks of $\mathbf{B}$ is calculated by summing the number of bits generated across all of the MAC-layer slots that fall in that subblock. On Line 9, the structure of the connection weights of the forecaster of the i'th device of D at the current level $L - \mathbf{B}.\text{depth}()$ is concatenated to $\mathbf{w}$ along the first dimension[7] of $\mathbf{w}$.

Fig. 4.3 shows the pseudo-code for an implementation of the ForecastFutureTraffic function. On Line 2, the duration of the subblock $\mathbf{b}$ of $\mathbf{B}$ is extracted. On Line 3, the matrix $\hat{\mathbf{x}}$ is initialized to the zero matrix whose number of rows equals the length of D, and whose number of columns equals the number of subblocks $|\mathbf{B}|/|\mathbf{b}|$ of block $\mathbf{B}$ at the current level of the MSA. For each of the devices in vector D, on Line 5, the input of the

[7]The "1" that appears as the third argument of the Concat function on Line 9 indicates that the concatenation is performed along the first dimension.

```
    ForecastFutureTraffic:
1   2D-matrix ForecastFutureTraffic($D, \mathbf{w}, \mathbf{x}, \mathbf{B}$) {
2    $|\mathbf{b}| = \mathbf{B}$.GetSubblockDuration();
3    $\hat{\mathbf{x}} = \text{zeros}(\text{length}(D), |\mathbf{B}|/|\mathbf{b}|)$;
4    for($i' = 0; i' < \text{length}(D); i'{+}{+}$) {
5     x_ForecasterInput = [ ];
6     $b = b_{\text{last}}^{D[i']}$;
7     while(length(x_ForecasterInput) $< n_0^{D[i']}$) {
8      x_ForecasterInput = [x_ForecasterInput, $\mathbf{x}[i', b]$];
9      $b = b - \lceil T_{D[i']}/|\mathbf{b}| \rceil$;
10    }
11    $\tilde{\mathbf{x}}$ = MLPForecaster($\mathbf{w}[i', :, :, :]$, x_ForecasterInput);
12    $b_{\text{last,temp}}^{i'} = b_{\text{last}}^{D[i']}$;
13    for($m = 1; m \leq K_{D[i']}; m{+}{+}$) {
14     $\tilde{b} = b_{\text{last}}^{D[i']} + \lceil m T_{D[i']}/|\mathbf{b}| \rceil$;
15     $b' = \tilde{b} - \text{size}(\mathbf{x}, 2)$;
16     if($b' < |\mathbf{B}|/|\mathbf{b}|$) {
17      $\hat{\mathbf{x}}[i', b'] = \tilde{\mathbf{x}}[m]$;
18      $b_{\text{last,temp}}^{i'} = \tilde{b}$;
19     } else break;
20    }
21    $b_{\text{last}}^{D[i']} = b_{\text{last,temp}}^{i'}$;
22   }
23   return $\hat{\mathbf{x}}$;
24  }
```

Figure 4.3. Pseudo-code for an implementation of the ForecastFutureTraffic function based on MLP

forecaster of the device with local index i', denoted by **x**_ForecasterInput, is initialized to the empty vector. We define b_{last}^{i} as the subblock in which the last traffic generation instance of device i occurred (in the entire past) before the current block **B**. (In this notation, i stands for the global index of the device, which was set at the beginning of this chapter and fixed. In contrast, i' is the local index as used in this pseudo-code. Hence, $i = D[i']$ says that i is the i'th element of vector D.) We define n_0^{i} for each i (where i is the global index of the device) as the required number of inputs (specified exogenously) for the forecaster of device i. Our immediate goal will be to find the indices of the last $n_0^{D[i']}$ subblocks (before the current block **B**) in which device $D[i']$ has generated traffic. To this end, we define b as a temporary variable that holds the (global) index of the last such subblock found so far. On Line 6, this variable b is initialized to $b_{\text{last}}^{D[i']}$. On Line 7, while the total number of elements of the vector **x**_ForecasterInput is less than $n_0^{D[i']}$, on Line 8, the number of bits generated by device $D[i']$ in block b, namely $\mathbf{x}[i',b]$, is concatenated to **x**_ForecasterInput. On Line 9, b is updated to the subblock that contains the last generation instance of device $D[i']$. (Recall that T_i is defined as the period with which device i generates traffic.) We define the temporary variable $\tilde{\mathbf{x}}$ as the vector of the forecast number of bits of the device with local index i'. The function MLPForecaster, which appears on Line 11, is our MLP-based forecaster, whose pseudo-code appears in Fig. 4.9 in Section 4.4.2. On Line 11, this function computes $\tilde{\mathbf{x}}$. On Line 12, the temporary variable $b^{i'}_{\text{last,temp}}$ is set equal to $b^{D[i']}_{\text{last}}$. On Line 14, $\tilde{b}$ denotes the global index of the subblock in which the mth generation instance of device $D[i']$ will fall in the future. On Line 15, b', which is the local index of the subblock within block $\mathbf{B}$, is computed by subtracting the number of past subblocks from $\tilde{b}$. On Lines 16-18, if the local index of the subblock is less than the total number of subblocks in block $\mathbf{B}$, the forecast traffic generation pattern for the device with local index i' in subblock b' is set to the value of the mth step ahead prediction (for the device with local index i'), and $b^{i'}_{\text{last,temp}}$ is updated to the global index of the subblock that contains the latest forecast generation instance of the device with local index i'. Otherwise, on Line 19, marking the subblocks that can have a positive number of bits of forecast traffic for the device with local index i' has been completed; hence, the function breaks out of the *for* loop. On Line 21, $b^{D[i']}_{\text{last}}$ is set equal to $b^{i'}_{\text{last, temp}}$. On Line 23, the function returns the 2D matrix $\hat{\mathbf{x}}$ of the forecast traffic generation patterns of all of the devices in vector D at the level of the MSA at which the ForecastFutureTraffic function has been called.

We now discuss a particular implementation of the ScheduleDevicesAtMACLayer function that we employ in this chapter. The pseudo-code of this implementation appears in Fig. 4.4. The $\mathbf{a}$, $\mathbf{p}$, $\mathbf{r}$, and $\mathbf{d}$ are the vectors of a_j, p_j, r_j and d_j, respectively, over the set of bursts that appear over the current scheduling window at the MAC layer. On Line 2, each of these vectors is initialized to null. On Lines 3 - 13, the double *for* loop iterates over all of the devices in vector D (Line 3) and over all of the MAC-layer slots within the scheduling window (Line 4). On Line 5, if the number of bits for the device with local index i' in slot k' is positive, this indicates that a burst is generated in this slot. (Recall our assumption in Section 1.4 that each device generates at most a single burst in any MAC-layer slot.) On Line 6, the number of bits in the k'th time slot of $\hat{\mathbf{x}}$ for device i is concatenated to vector $\mathbf{a}$. On Line 7, the processing time of burst j is calculated as the smallest integer greater than or equal to the number of bits in the k'th time slot of $\hat{\mathbf{x}}$ for the device with local index i' divided by $C_{D[i']}$, and appended to $\mathbf{p}$. On Line 8, the k'th time slot is concatenated to vector $\mathbf{r}$. Based on our assumption

```
     ScheduleDevicesAtMACLayer:
1    2D-matrix ScheduleDevicesAtMACLayer($D, \hat{\mathbf{x}}$) {
2     $\mathbf{a} = [\,]; \mathbf{p} = [\,]; \mathbf{r} = [\,]; \mathbf{d} = [\,]; \mathbf{f} = [\,];$
3     for($i' = 0; i' < \text{length}(D); i'$++) {
4      for($k' = 0; k' < \text{size}(\hat{\mathbf{x}}, 2); k'$++) {
5       if($\hat{\mathbf{x}}[i', k'] > 0$) {
6        $\mathbf{a} = [\mathbf{a}, \hat{\mathbf{x}}[i', k']];$
7        $\mathbf{p} = [\mathbf{p}, \lceil \hat{\mathbf{x}}[i', k'] / C_{D[i']} \rceil];$
8        $\mathbf{r} = [\mathbf{r}, k'];$
9        $\mathbf{d} = [\mathbf{d}, \min(k' + \lceil \Delta^{D[i']} / \tau_{MAC} \rceil - 1, \text{size}(\hat{\mathbf{x}}, 2) - 1)];$
10       $\mathbf{f} = [\mathbf{f}, D[i']];$
11      }
12     }
13    }
14    $\mathbf{Y} = \text{JSTW}(\mathbf{a}, \mathbf{p}, \mathbf{r}, \mathbf{d});$
15    $\mathbf{S} = \text{AssignSlotsToDevices}(\mathbf{Y}, \mathbf{f});$
16    return $\mathbf{S}$;
17   }
```

Figure 4.4. Pseudo-code for an implementation of the ScheduleDevicesAtMACLayer function

```
     AssignSlotsToDevices:
1    2D-matrix AssignSlotsToDevices($\mathbf{Y}, \mathbf{f}$) {
2     $\mathbf{S} = \text{zeros}(N, \text{size}(\mathbf{Y}, 2));$
3     for($j' = 0; j' < \text{length}(\mathbf{f}); j'$++) {
4      $i = \mathbf{f}[j'];$
5      $\mathbf{S}[i, :] = \mathbf{S}[i, :] + \mathbf{Y}[j', :];$
6     }
7     return $\mathbf{S}$;
8    }
```

Figure 4.5. Pseudo-code for an implementation of the AssignSlotsToDevices function

(Section 1.4) that each IoT device is associated with a single application in this chapter, we define the delay constraint of the device that generated burst j as $\Delta^{f(j)} \equiv \Delta_j$. On Line 9, the deadline of burst j is calculated and is concatenated to the vector $\mathbf{d}$. (The minimum taken on this line treats any deadline beyond the current scheduling window as if it were at the end of this scheduling window.) On Line 10, the i'th device in vector D, which generated burst j, is concatenated to the vector of devices $\mathbf{f}$.

On Line 14, the schedule for all of the bursts within the current scheduling window is computed by solving JSTW via any mixed-integer solver. (We emphasize that any collision-free burst scheduling may be used in the place of JSTW.) On Line 15, the

MAC-layer slots that are allocated to devices are calculated based on the slots allocated to the bursts of these devices (which appear in **Y**).

The AssignSlotsToDevices function appears in Fig. 4.5. On Line 2, the scheduling matrix **S** is initialized as the zero matrix. The entry (i,k) of the **S** matrix is 1 if slot k is allocated to device i and is 0 otherwise. On Line 4, for each burst with local index j', the index of the device, denoted by i, that generates j' is obtained from the vector **f**. On Line 5, the schedule of burst j' is added onto the schedule for device i.

```
    BalanceLoad:
1   [seqOfVectors, seqOfBlocks] BalanceLoad($D, \hat{\mathbf{x}}, \mathbf{B}$) {
2    $\mathbf{p} = [\,]; \mathbf{r} = [\,]; \mathbf{d} = [\,];$
3    $\mathbf{f} = [\,]; \langle\tilde{D}\rangle = [\,];$
4    $|\mathbf{b}| = \mathbf{B}$.GetSubblockDuration();
5    for($i' = 0; i' <$ length($D$); $i'$++) {
6     for($b' = 0; b' <$ size($\hat{\mathbf{x}}, 2$); $b'$++) {
7      if($\hat{\mathbf{x}}[i', b'] > 0$) {
8       $\mathbf{p} = [\mathbf{p}, \lceil \hat{\mathbf{x}}[i', b'] / C_{D[i']} \rceil];$
9       $\mathbf{r} = [\mathbf{r}, b'];$
10      $\mathbf{d} = [\mathbf{d}, \min(b' + \lceil \Delta^{D[i']} / |\mathbf{b}| \rceil - 1, \mathrm{size}(\hat{\mathbf{x}}, 2) - 1)];$
11      $\mathbf{f} = [\mathbf{f}, D[i']];$
12     }
13    }
14   }
15   $\mathbf{Z} = \mathrm{LBB}(\mathbf{p}, \mathbf{r}, \mathbf{d});$
16   for($j' = 0; j' <$ size($\mathbf{Z}, 1$); $j'$++) {
17    for($b' = 0; b' <$ size($\hat{\mathbf{x}}, 2$); $b'$++) {
18     if($\mathbf{Z}[j', b'] == 1$ && $\mathbf{f}[j'] \notin \langle\tilde{D}\rangle[b']$)
19       $\langle\tilde{D}\rangle[b'] = [\langle\tilde{D}\rangle[b'], \mathbf{f}[j']];$
20    }
21   }
22   $\langle\mathbf{b}\rangle = \mathbf{B}$.GetAllSubblocksInTemporalOrder();
23   return $[\langle\tilde{D}\rangle, \langle\mathbf{b}\rangle]$;
24  }
```

Figure 4.6. Pseudo-code for the BalanceLoad function

The pseudo-code of the BalanceLoad function appears in Fig. 4.6. On Lines 2-3, the function initializes each of the processing time **p**, generation time **r**, delay deadline **d** and the device index **f** vectors as well as the sequence $\langle\tilde{D}\rangle$ of device vectors to null. On Line 4, the duration of a subblock of **B** is extracted from the structure **B**. For each device i' (Line 5) and for each subblock b' (Line 6), if the number of bits generated by device i' in subblock b', namely $\hat{\mathbf{x}}[i', b']$, is positive (i.e. if there is a job at this level), the processing time of that job is calculated and concatenated to $\mathbf{p}$ on Line 8. On Line 9, the subblock index b' is concatenated to $\mathbf{r}$. On Line 10, the delay deadline of the job is calculated (setting the delay deadline to the end of the current block $\mathbf{B}$ if the actual delay deadline is some time after the current block) and is concatenated to the vector $\mathbf{d}$. On Line 11, the i'th device in vector D is concatenated to the vector of devices $\mathbf{f}$. On Line 15, the LBB optimization program for load balancing is solved and returns the matrix $\mathbf{Z}$. (The entry (j', b') of $\mathbf{Z}$ is the number of bits of job j' that is assigned at the output of load balancing to subblock b'.) On Line 18, if the bits of job j' have been loaded onto subblock b' and the device $\mathbf{f}[j']$ that generated the job j' has not yet been added to $\langle\tilde{D}\rangle[b']$, namely the device vector for subblock b', that device is concatenated to $\langle\tilde{D}\rangle[b']$. On Line 23, the function returns the sequence $\langle\tilde{D}\rangle$ of device vectors over the temporally ordered sequence $\langle\mathbf{b}\rangle$ of subblocks of $\mathbf{B}$ as well as the sequence $\langle\mathbf{b}\rangle$ itself.

In Fig. 4.7, we illustrate the operation of MSA on a single scheduling window $[0, T_{sch}^{(L)}]$. The total number of levels is $L+1 = 3$. We begin at Level 2 with the scheduling window $[0, T_{sch}^{(2)}]$ and $M_2 = 2$. Let $R_b^{(l)}$ denote subblock b of Level l. On Line 7 of Fig. 4.1, when MSA balances the forecast load, in this example, it produces $D^{(R_1^{(2)})} = \{1,4,6,7\}$ as the set of devices that have a positive number of forecast bits in subblock $R_1^{(2)}$ and $D^{(R_2^{(2)})} = \{1,2,3,5,7\}$ as that in subblock $R_2^{(2)}$. Now, on Line 9 of Fig. 4.1, MSA is invoked for subblocks $R_1^{(1)}$ and $R_2^{(1)}$. For $R_1^{(1)}$ and $R_2^{(1)}$, forecasting needs to be performed only for those $i \in D^{(R_1^{(2)})}$, not for the entire $\mathscr{N}$. On Line 3 of Fig. 4.1, this forecasting produces $\{1,6,7\}$ and $\{1,4,6,7\}$ as the device indices with a positive number of forecast bits for subblocks $R_1^{(1)}$ and $R_2^{(1)}$, respectively. Invoking Line 9 on $R_1^{(1)}$ reaches the base case of the recursion and produces a MAC-layer schedule for devices in $\{1,6,7\}$ based on their forecast bursts at Level 0. As the MSA continues its recursion, whenever it reaches Level 0, the next MAC-layer schedule over a scheduling window of length $T_{sch}^{(0)}$ is created. The key point of MSA is that since accurate forecasts are available for longer scheduling windows at the higher levels, balancing the load at those levels pre-selects the device sets that need to be considered over the much shorter scheduling window at Level 0. In this example, the MSA continues its recursion until all devices have been scheduled at the MAC layer on $[0, T_{sch}^{(2)}]$.

4.4 Results

We divide all IoT devices into four distinct device classes: (1) FBP (2) VBP, (3) FBA, and (4) VBA. Here, "Fixed Bit" indicates that the device generates a constant number of bits at all its traffic generation times, and "Variable Bit" indicates otherwise.

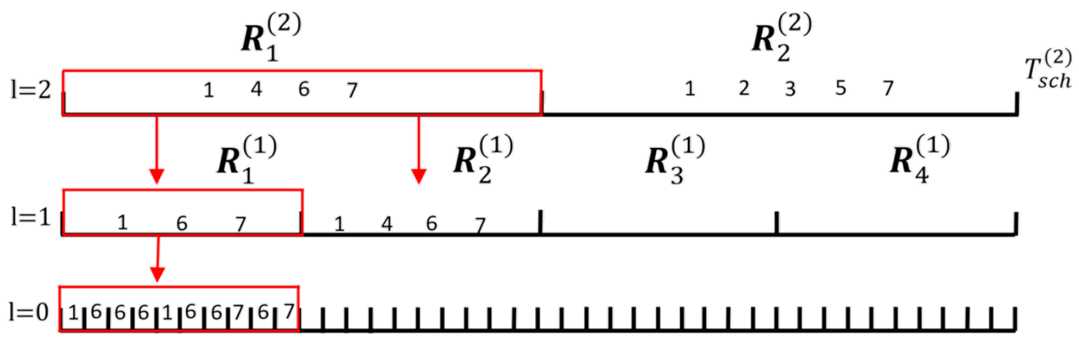

Figure 4.7. Operation of the Multi-Scale Algorithm

Furthermore, "Periodic" indicates that the device generates its traffic at regular intervals, and "Aperiodic" indicates otherwise.

The reason for this classification is three-fold: First, if we represent the traffic generation instances of the FBA traffic in discrete time, forecasting an FBA traffic pattern is equivalent to a binary classification problem at each discrete time instance: The forecaster needs to decide only whether a fixed-bit burst was generated at that time instance or not. In contrast, forecasting VBA traffic requires, in addition, the estimation of the number of bits at each generation instance. Second, forecasting VBP traffic can utilize the knowledge that a burst is generated at every period of the traffic pattern. In contrast, forecasting VBA traffic requires, in addition, the estimation of the inter-arrival time between successive generation instances. Third, the traffic generation pattern of a device that falls in the FBP class requires no forecasting, once the generation time and the number of bits of any burst of this device is known. Thus, each IoT device class corresponds to the type of forecasting problem that must be solved. Our classification is a reflection of the differences among these forecasting problems.

4.4.1 Data Collection and Processing Methodology

First, we obtained actual measurements from (1) DHT22 (Temperature, Relative Humidity (RH)), LDR and MQ135 (Air Quality) sensors in our laboratory (collected over 2.5 months), and (2) the Air Quality measurements in (De Vito et al., 2008)(University of California Irvine Machine Learning Repository, 2008) (collected over 1 year). These measurements fall in the VBP, FBA, and VBA classes. Second, we classified the reporting of the sensor measurements into two types: those reported (A) via data transmission at regular intervals, and (B) in an event-triggered fashion, i.e., only if the measurement changes by more than a threshold compared with the previous measurement. (The former falls under “Periodic” and the latter falls under “Aperiodic” device classes.) In this study, we assumed that every IoT device is associated with a single sensor. (Our algorithm in Section 4.3.2 does not rely on this assumption.) Third, after this classification, we represented each reading from a given sensor by a fixed number of bits. Fourth, we compressed the resulting stream of fixed-bit readings from each sensor via Huffman coding on the entire stream. (We plan to investigate other compression schemes in our future work.) The traffic generation pattern of each IoT device in the VBP, FBA and VBA classes is thus this compressed stream of bits. Finally, we simulated traffic generation patterns for smart meter readings, which fall in the FBP class, by generating a single burst with a constant number of bits per device period (which is 1 hour in our case) for each meter. The initial traffic generation time is uniformly distributed over such a period and is independent across devices. None of the bitstreams generated for the FBP class is compressed.

In order to create a simulation environment with a large number N of IoT devices, we bootstrapped from the above representative IoT devices that belong to each of the four distinct device classes. For the FBP and VBP classes, we performed bootstrapping by aligning the generated patterns over all periods and selecting a sample at random from each time point. For the FBA and VBA classes, we computed the empirical autocorrelation function of the compressed traffic generation pattern of each IoT device. For each such IoT device i, this pattern typically has multiple periods, and we selected the largest period $T_i^{(max)}$ of this pattern. Then, we divided the traffic generation pattern of device i into segments of length $T_i^{(max)}$, aligned these segments, and formed the traffic generation pattern of each bootstrapped device by selecting at random from samples that are aligned at the same point in each period. This procedure is aimed at preserving the inter-temporal characteristics of the data within each segment in forming bootstrapped traffic generation patterns for the FBA and the VBA classes.

4.4.2 Choice and Parameters of the Forecasting Scheme

We have carried out the forecasting performed by Gateway G for each device in the bootstrapped dataset. We formed forecasts only for the VBP, FBA and VBA classes. The optimization of a MLP neural network (whose MSE was minimized using the LM algorithm (Haykin, 2010)) that is trained for each IoT device in MATLAB was observed to achieve a high throughput for the Multi-Scale Algorithm, compared with RAB (Section 4.4.4). Furthermore, the execution time per device was less than 1 second (see Section 4.4.4), which is sufficient for the devices in our dataset. For each IoT device, the training times for MLP ranged from 30 seconds to 2 minutes for approximately 10,000 samples with a storage requirement of approximately 2 kB. (Thus, training 10,000 devices at G has reasonable time and space complexity requirements.) Under the network simulation set-up of Section 4.4.3, we have empirically validated for MLP that $\tilde{T}_{il}^{af}$, determined based on MSE, is a monotonically increasing function of l for each i. For these reasons, the MLP is used as the forecasting scheme in this work.

In order to forecast the traffic of all IoT devices in the VBP, FBA and VBA classes, we use a variable learning rate in the LM algorithm. We set the initial learning rate to 0.001. Furthermore, we set the decrease and increase factors to 0.1 and 10, respectively, and the maximum learning rate to 10^{10}. In the design of the forecaster MLP, invoked for a given device at a given level of the MSA, we use the first 70% of the set of samples for training, the next 15% for validation, and the final 15% for testing. All of the hidden-layer neurons utilize the *tanh* activation function while the output neurons are kept linear. The number of outputs of the MLP is $\lceil T_{sch}^{(l)}/B_l \rceil$ at level l. We let n_0^i denote the number of samples into the past that will be used for forecasting the traffic of device i. (Each such past sample is fed as an input to the MLP.) For each $n_0^i : 1 \leq n_0^i \leq 100$, we run the MLPTraining function, which shall be described next, and pick the optimal n_0^i, denoted by $(n_0^i)^*$, that results in the lowest sMAPE over all choices of n_0^i in this range. (The optimal values $(n_0^i)^*$ for a set of representative IoT devices will be displayed in Table 4.1.)

Fig. 4.8 displays the offline training algorithm for the MLP, which computes the connection weights for each device i. The calling script for the MLPTraining function appears on Lines 1-19, and the MLPTraining function itself is shown on Lines 20-26. We execute the calling script for the MLPTraining function only once before we run the MSA on the entire sequence of scheduling windows that appear at the top level L.[8] On Lines 1-19, the calling script iterates through each level l (Line 1) and each device i (Line 2) in the entire set of devices $\mathscr{N}$. We now describe the operations that are executed (Lines 3-17) for each l and each i. We denote by $\mathbf{X}_{\text{init}}$ the past traffic generation pattern before training begins. On Line 3, the total number of subblocks that have passed until training begins, which is denoted by *nInitSubblocks*, is computed. On Line 4, the matrix $\tilde{\mathbf{X}}$ is formed by reshaping (as the term is used in MATLAB) $\mathbf{X}_{\text{init}}[i,:]$ such that the number of rows of the reshaped matrix $\tilde{\mathbf{X}}$ equals the number of MAC-layer slots in a subblock at level l, and its number of columns equals the number of past

[8]In practice, a refresh period may be chosen to re-train the MLP such that it incorporates into its connection weights the new traffic generation patterns that form as MSA operates on successive scheduling windows.

```
    Calling Script for MLPTraining:
1   for($l = 0; l \leq L; l$++) {
2    for($i = 0; i < N; i$++) {
3     $nInitSubblocks = \text{size}(\mathbf{X}_{\text{init}}, 2)/(\lceil B_l/\tau_{MAC} \rceil)$;
4     $\tilde{\mathbf{X}} = \text{Reshape}(\mathbf{X}_{\text{init}}[i,:], \lceil B_l/\tau_{MAC} \rceil, nInitSubblocks)$;
5     $\mathbf{x} = \text{Sum}(\tilde{\mathbf{X}}, 1)$;
6     $nSamples = \text{length}(\mathbf{x}) - (n_0^i + K_i)$;
7     $\mathbf{X}_{\text{MLP}} = \text{zeros}(nSamples, n_0^i)$;
8     $\mathbf{Y}_{\text{MLP}} = \text{zeros}(nSamples, K_i)$;
9     for($m = 0; m < nSamples; m$++) {
10     $\mathbf{X}_{\text{MLP}}[m,:] = \mathbf{x}[m : (m + n_0^i - 1)]$;
11     $\mathbf{Y}_{\text{MLP}}[m,:] = \mathbf{x}[(m + n_0^i) : (m + n_0^i + K_i - 1)]$;
12    }
13    $\mathbf{X}_{\text{train}} = \mathbf{X}_{\text{MLP}}[0{:}\lceil 0.7\ nSamples \rceil - 1, :]$;
14    $\mathbf{Y}_{\text{train}} = \mathbf{Y}_{\text{MLP}}[0{:}\lceil 0.7\ nSamples \rceil - 1, :]$;
15    $\mathbf{X}_{\text{val}} =$
         $\mathbf{X}_{\text{MLP}}[\lceil 0.7\ nSamples \rceil{:}\lceil 0.85\ nSamples \rceil - 1, :]$;
16    $\mathbf{Y}_{\text{val}} =$
         $\mathbf{Y}_{\text{MLP}}[\lceil 0.7\ nSamples \rceil{:}\lceil 0.85\ nSamples \rceil - 1, :]$;
17    $\mathbf{W}[l, i, :, :, :] =$
         $\text{MLPTraining}(\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}, \mathbf{X}_{\text{val}}, \mathbf{Y}_{\text{val}}, n_0^i, K_i)$;
18   }
19  }
    MLPTraining:
20  3D-matrix $\text{MLPTraining}(\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}, \mathbf{X}_{\text{val}}, \mathbf{Y}_{\text{val}}, n_0^i, K_i)$ {
21   $\tilde{E} = 3; \tilde{N} = 1$;
22   $mlp = \text{MLP\_SSA}(\mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}, \mathbf{X}_{\text{val}}, \mathbf{Y}_{\text{val}}, n_0^i, K_i, \tilde{E}, \tilde{N})$;
23   $\text{TrainMLP}(mlp, \mathbf{X}_{\text{train}}, \mathbf{Y}_{\text{train}}, \mathbf{X}_{\text{val}}, \mathbf{Y}_{\text{val}})$;
24   $\mathbf{w} = mlp \rightarrow \text{connectionWeights}()$;
25   return $\mathbf{w}$;
26  }
```

Figure 4.8. Pseudo-code for the MLPTraining function

subblocks (until training begins). On Line 5, the number of bits generated in each such past subblock is calculated by summing the number of bits generated across all of the MAC-layer slots that fall in that subblock. On Line 6, the total number samples *nSamples* that will be used for training the MLP is calculated. We assume that the value of *nSamples* is greater than or equal to 100 throughout this pseudo-code. On Line 7, the input matrix for the MLP, namely $\mathbf{X}_{\text{MLP}}$, is set to the zero matrix, whose number of rows is *nSamples* and whose number of columns is the total number of the inputs of MLP, which is n_0^i. On Line 8, the desired output matrix for the MLP, namely $\mathbf{Y}_{\text{MLP}}$, is set to the zero matrix, whose number of rows is *nSamples* and whose number of columns is

K_i. On Lines 9-12, for each sample m, $\mathbf{X}_{\text{MLP}}[m,:]$ is set equal to the traffic generation pattern that consists of n_0^i samples starting at time m (Line 10), and $\mathbf{Y}_{\text{MLP}}[m,:]$ is set equal to the traffic generation pattern that consists of K_i samples starting at $m+n_0^i$ (Line 11). On Lines 13-14, each of $\mathbf{X}_{\text{train}}$ and $\mathbf{Y}_{\text{train}}$ is set to the first 70% of each of the $\mathbf{X}_{\text{MLP}}$ and $\mathbf{Y}_{\text{MLP}}$, respectively. Furthermore, on Lines 15-16, each of $\mathbf{X}_{\text{val}}$ and $\mathbf{Y}_{\text{val}}$ is set to the next 15% of each of the $\mathbf{X}_{\text{MLP}}$ and $\mathbf{Y}_{\text{MLP}}$, respectively. On Line 17, the 3D matrix of connection weights of device i at level l, namely $\mathbf{W}[l,i,:,:,:]$, is set equal to the output of the MLPTraining function.

On Line 22 of Fig. 4.8, the MLPTraining function calls the MLP structure selection algorithm, implemented by the MLP_SSA function in Fig. 2.1 (in Section 2.2.2), which returns a pointer to the selected MLP architecture. On Line 23, the local optimal MLP architecture is trained with the training data. (As noted earlier, we used the LM algorithm as a specific implementation of TrainMLP.) On Lines 24 and 25, $\mathbf{w}$ is set to the 3D matrix of connection weights of the MLP architecture and returned. (The entry (e,n,m) of the $\mathbf{w}$ is the connection weight from neuron m at layer $e-1$ to neuron n at layer e.)

Table 4.1. The number of past samples used and the local-optimal MLP architecture for representative IoT devices

Sensor	Class	$(n_0^i)^*$	MLP Architecture $(n_1, n_2, \ldots, n_E)$
Temperature	VBP	30	(1, 5)
Relative Humidity	VBP	5	(15, 8, 5)
LDR	VBP	10	(5, 1, 5)
Air Quality	VBA	20	(2, 5)
NO_2	VBA	3	(9, 20, 5)
NHMC	FBA	3	(8, 5, 5)
CO	FBA	30	(4, 5)

In our simulations, the resulting forecaster MLPs had at most 2 hidden layers, each of which contained at most 20 neurons, while the number of neurons in each layer ranged from 3 to 30. Table 4.1 displays the information on the MLP architecture found for a representative set of IoT devices, organized into the following columns: (1) the sensor that is associated with the IoT device, (2) the traffic class of the IoT device, (3) the local optimal number of inputs to the MLP, denoted by $(n_0^i)^*$, that resulted in the MLP architecture (found by MLP-SSA) in the lowest $\text{sMAPE}_{\text{allLayers}}^{\text{lowest}}$ (Fig. 2.1) over all $n_0^i \in [1,100]$, and (4) the corresponding MLP architecture. In the fourth column of Table 4.1, each MLP architecture is described by a row vector, each entry of which shows the number of neurons n_e in each layer $e \in \{1,\dots,E\}$, where E denotes the number of layers. The first hidden layer appears as the leftmost and the output layer appears as the rightmost entry of this row vector. The number of neurons in the output layer of each MLP architecture was fixed at 5 due to the empirical finding that appears at the end of Section 4.4.3.

```
    MLPForecaster:
1   vector MLPForecaster($\mathbf{w}_i, \mathbf{x}_i$) {
2     $\tilde{\mathbf{x}} = \mathbf{x}_i$;
3     $E = \text{size}(\mathbf{w}_i, 1)$;
4     for($e = 1; e \leq E; e$++) {
5       $\tilde{\mathbf{x}} = [\tilde{\mathbf{x}}, 1]$;
6       if($e == E$) $\tilde{\mathbf{x}} = \mathbf{w}_i[e,:,:]\tilde{\mathbf{x}}$;
7       else $\tilde{\mathbf{x}} = tanh(\mathbf{w}_i[e,:,:]\tilde{\mathbf{x}})$;
8     }
9     return $\tilde{\mathbf{x}}$;
10  }
```

Figure 4.9. Pseudo-code for the MLPForecaster function

After the training of the MLP for each device has been completed offline, as described in this section so far, the MLP is used for online forecasting, which we describe next: Fig. 4.9 displays the online MLP-based forecasting scheme for device i. The first input to the MLPForecaster function, namely $\mathbf{w}_i$, is a 3D matrix of connection weights for device i, whose entry (e,n,m) is the connection weight that emanates from neuron m at layer $e-1$ and enters into neuron n at layer e. The last column of $\mathbf{w}_i[e,:,:]$ is the bias term of layer e. The second input of MLPForecaster, namely $\mathbf{x}_i$, is the vector of the past traffic generation pattern of device i for the level for which the MLPForecaster is invoked. The vector $\tilde{\mathbf{x}}$ is a temporary vector that shall denote the output vector of each neural layer. On Line 2, $\tilde{\mathbf{x}}$ is initialized to $\mathbf{x}_i$ at the input layer. On Line 3, the number of hidden layers E is set to the size of $\mathbf{w}_i$ along the first dimension. Lines 4-8 calculate the output vector $\tilde{\mathbf{x}}$ of the MLP-based neural network. On Line 5, the bias multiplier is concatenated onto $\tilde{\mathbf{x}}$. On Line 6, if the current layer is the last layer of the MLP, in order to compute the output of layer e, $\tilde{\mathbf{x}}$ is multiplied by the matrix of connection weights. Otherwise, on Line 7, the tangent hyperbolic activation function is applied to the product of the connection weights and $\tilde{\mathbf{x}}$. On Line 9, the function returns the output vector of the MLP.

4.4.3 Network Simulation Set-up

In line with Sections 1.4 and 4.2, we set up a network simulation whose aim is to schedule the bursts generated by N devices at the MAC layer. Our simulation does not use the traditional TCP/IP protocol stack, but rather schedules the bursts generated by the IoT applications (specified in Section 4.4.1) directly at the MAC layer.

We present results for two network simulation set-ups: First, in Section 4.4.4, we measure the performance of the MSA under a scenario in which the devices generate a relatively heavy traffic load with respect to the total system capacity. Second, in Section 4.4.5, we measure the performance of MSA under the IEEE 802.11ah standard ("IEEE Standard for Information technology–Telecommunications and information exchange between systems - Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub 1 GHz License Exempt Operation", 2017), where the traffic generated by the set of IoT devices in our simulations turns out to be relatively light compared with the available system capacity.

The set of delay constraints (in seconds) of all of the devices in our simulations is as follows[9]: $\Delta^{(1)} = 0.5$, $\Delta^{(2)} = 1$, $\Delta^{(3)} = 2$, $\Delta^{(4)} = 180$, $\Delta^{(5)} = 600$ and $\Delta^{(6)} = 3600$ seconds, with the corresponding traffic generation intervals $T_1 = T_2 = 180$, $T_3 = 3600$, $T_4 = 180$, $T_5 = T_6 = 3600$ seconds. Since $T_{min} = 180$ seconds, after eliminating the empty bins (Section 4.3.2), $\tilde{\Delta}_1 = 0.5$, $\tilde{\Delta}_2 = 600$, $\tilde{\Delta}_3 = 3600$ seconds. Thus, the total number of levels is $L+1 = 3$. We assume that $\tau_{MAC} = 0.1$ second. Thus, this set-up falls under Case 2 in Section 4.3.2. Hence, $B_0 = 0.1$, $B_1 = 600$, and $B_2 = 3600$ seconds. We found empirically that $K_i = 5$ for all i (in the relation $\tilde{T}_{il}^{af} = K_i T_i$) is the approximate point at which the MSE is minimized for each i and each l. Thus, $T_{sch}^{(0)} = 900$, $T_{sch}^{(1)} = 3000$, and $T_{sch}^{(2)} = 18000$ seconds.

In Section 4.4.4, the MAC-layer slot capacities that we utilize is given by the sequence $C^{(1)} = 10$, $C^{(2)} = 12$, $C^{(3)} = 10$, $C^{(4)} = 15$, $C^{(5)} = 15$, and $C^{(6)} = 128$ kbits per slot. In Section 4.4.5, the MAC-layer slot capacities are determined based on the set of data rates under the 802.11ah standard and their calculation will be detailed in that section.

[9]We use a superscript on the delay constraints to denote the elements of the sequence of these delay constraints in order to distinguish them from the notation Δ_j, which stands for the delay constraint for *burst j*.

4.4.4 Results on Joint Forecasting-Scheduling

Even though the performance of different forecasting schemes with respect to the sMAPE metric was evaluated in Chapter 2, we have found that sMAPE, measured at the output of the forecaster, does not correlate well with network performance at the output of the scheduler in all instances. For example, sMAPE does not account for the fact that the impact of the underallocation of burst processing times on network throughput is different from that of overallocation. (While underallocation causes a burst not to be delivered, overallocation permits the delivery of that burst while possibly preventing the scheduling of adjacent bursts.) In general, the traditional forecasting performance metrics, such as sMAPE, are not specific to the application that will utilize those forecasts. In this work, rather than attempt to construct new forecasting metrics that correlate well with network performance, we report directly the network performance obtained by joint forecasting-scheduling.

To this end, in this section, we compare the uplink cross-layer throughput η (as defined in Section 1.4) and the transmit energy consumption of the MSA with those of the following three protocols: (1) RAB, which is a reactive, joint access-scheduling protocol that we have designed that combines the features of Access Class Barring (Ghavimi & Chen, 2015)(Leyva-Mayorga, Tello-Oquendo, Pla, Martinez-Bauset, & Casares-Giner, 2016) and reservation-based scheduling (Yu et al., 2013), (2) PAL, which is a proactive heuristic that we have designed that utilizes non-preemptive priority scheduling (Stallings, 2009) such that the priority is determined based on the time-averaged traffic load until the delay deadline, and (3) E-PRV-BO protocol, which is an enhancement that we have designed on PRV-PO (Hammad et al., 2017). We shall also compare the network performance of joint forecasting-scheduling with that obtained under perfect forecasts.

First, RAB is a protocol that we have developed that reacts to current traffic demand and uses a parameter p_{RAB} similar to the p_{ACB} parameter of the ACB protocol of LTE[10]. In RAB, there is no separate access channel. (Recall that a single channel is assumed throughout this chapter.) At each point in time, a device i may in one of four states: transmit, idle, backoff, and wait. If there is any traffic in i's queue in the idle state, i starts sending its data with probability p_{RAB} in the first upcoming slot or defers its access attempt to the next slot with probability $1 - p_{RAB}$. The additional number of slots, denoted by O_i, for which i wants to continue to send its data, is piggybacked

[10]The performance of ACB on the PRACH would give only the performance of the access channel, whereas in this chapter, we focus on the network performance of both the access and the data channels; hence, we do not compare directly with ACB.

by i to the data sent in each slot. In RAB, G acknowledges the reception of data on each slot to i and we allow this feedback to be instantaneous to give full advantage to RAB. Whenever the Acknowledgment (ACK) is not received (which may only be due to an uplink collision in our simulations), i goes into backoff with a global exponential backoff parameter λ per slot, which is identical for all devices. During backoff, i drops any burst whose delay deadline has expired. On the other hand, whenever an ACK has been received, it signifies to i that the channel has been reserved for i for the next O_i slots, which was communicated to G on i's last transmission. On the downlink, G communicates (instantaneously) to all other devices that the channel has been reserved for the next O_i slots, upon which all other devices go into the wait state. Upon exiting the wait state, the device goes into the idle state. In addition, any device whose backoff has expired goes into the wait state if the current slot has been reserved and goes into the idle state otherwise. (Note that there is no Carrier Sensing in RAB.) In summary, RAB combines the features of ACB and reservation-based access protocols to provide efficient utilization of a single uplink channel.

Second, PAL is a proactive heuristic protocol that we have developed, which allocates the single channel without preemption to the burst with the highest forecast "average load", defined as the forecast number of bits in burst j divided by the remaining number of slots until the deadline d_j. Similar to MSA, each device is communicated on the downlink in advance the slots allocated to its next forecast burst, as decided by PAL.

Third, PRV-PO (Hammad et al., 2017) is a QoS-aware *downlink* scheduling heuristic that maximizes the energy efficiency (at the base station) of heterogenous traffic that is comprised of delay-constrained, rate-constrained and best-effort classes. It assumes that the future channel state is available via a ray tracing engine; however, it does not forecast the (downlink) future traffic generation patterns of devices. We have modified PRV-PO for the *uplink*[11] multi-user channel in our case, as described below. Furthermore, we enhanced PRV-PO via the capability to forecast the future traffic generation patterns of individual IoT devices. The resulting protocol, called E-PRV-BO, has the following additional features: (1) The cross-layer uplink throughput η (defined in Section 1.4) replaces energy efficiency as the objective function.[12] (2) The number of connections of each IoT device is set to 1. (3) E-PRV-BO uses bursts in the place of packets in PRV-PO.

[11] In this modification, we assume that the queue length at each device is communicated to the IoT Gateway at the beginning of each frame. This would require an additional mechanism by which each device is scheduled for uplink access at the beginning of each frame.

[12] When η replaces energy efficiency as the objective function, E-PRV-BO achieves an energy consumption that is comparable to that of MSA (which will be shown in Fig. 4.12). We have set the objective function of E-PRV-BO to η in order to measure its maximum possible performance with respect to the main metric in this chapter.

Furthermore, in our simulations, all of the devices are delay-constrained. As opposed to the simulations in (Hammad et al., 2017), in which the only type of delay-constrained traffic is Voice-over-IP (VoIP), since IoT traffic has a wide range of delay constraints, we generalize the maximum delay D_{max} and the maximum jitter Δt_{max} parameters of PRV-PO to the delay constraint Δ_j and the maximum jitter Δt^j_{max} of burst j, respectively. In our simulations, we set the frame size of E-PRV-BO to $\max_{j \in \mathscr{J}} \Delta_j$, which is the longest delay constraint over all of the bursts on $[0, T^{(0)}_{sch}]$.[13]

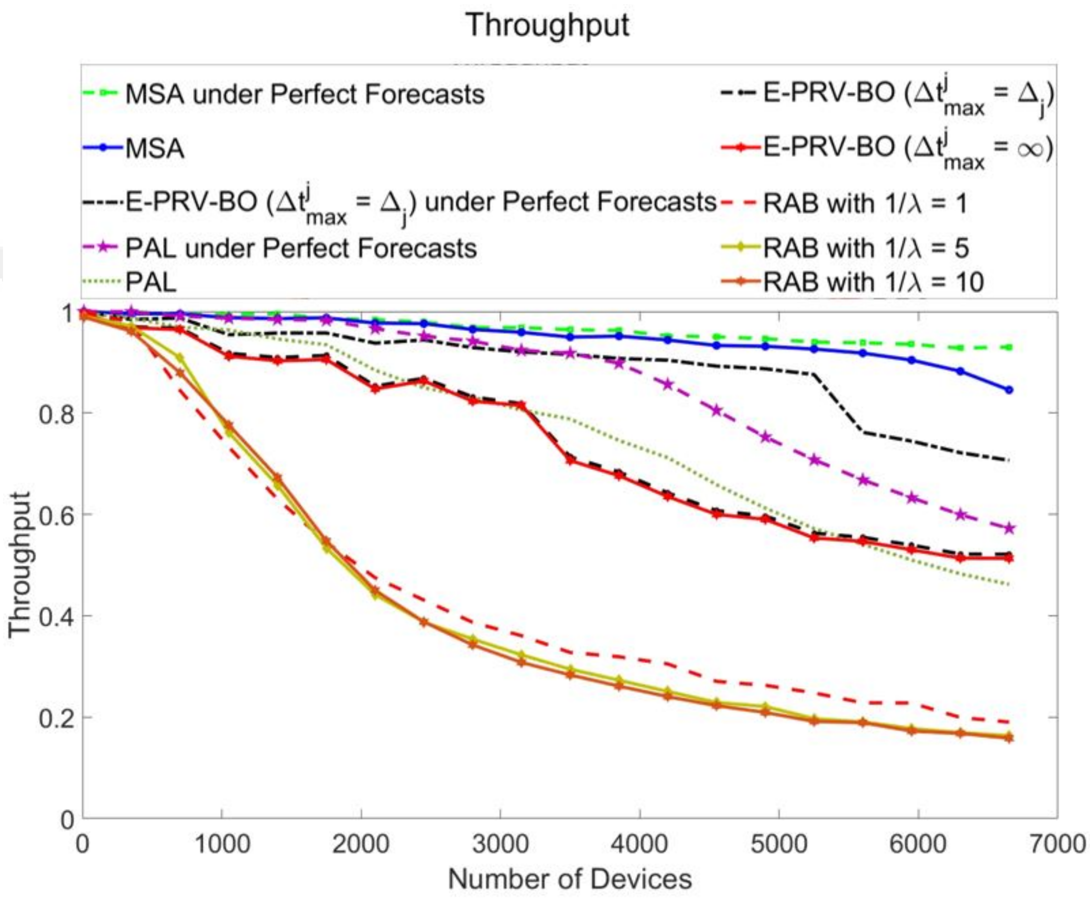

Figure 4.10. Throughput (as defined in Section 1.4) of the MSA compared with those of RAB, PAL, E-PRV-BO, and their respective upper bounds under perfect forecasts

We set up a simulation, in which the percentage of (bootstrapped) devices in each of the device classes (FBP, VBP, FBA, and VBA) is 25%. In Fig. 4.10, we present the results on the uplink cross-layer throughput η. First, for RAB, for each λ, we maximize η by picking p_{RAB} via exhaustive search. (The optimal values of p_{RAB} for representative

[13]This ensures that each burst that has not been scheduled in the current frame is handed over at most to the next frame.

λ's are displayed as a function of N in Fig. 4.11.) The resulting throughput is thus an upper bound on η for the dynamic version of RAB in which p_{RAB} would be adapted to network traffic. In Fig. 4.10, we see that the MSA significantly outperforms the performance upper bounds on dynamic RAB for all N for a wide range of λ's. (The control overhead of MSA, which will be quantified below, has already been incorporated into the simulation that produced Fig. 4.10.) By replacing Lines 2 and 3 in Fig. 4.1 with perfect forecasting, we obtain a performance upper bound on MSA. In Fig. 4.10, we see that the gap between the performance of MSA and this upper bound is very small and increases only gradually in N. Second, the MSA outperforms PAL, when Multi-Scale Forecasting is used as the forecasting scheme, as well as the upper bound to PAL obtained under perfect forecasts. The performance difference widens significantly beyond 3000 devices. Third, MSA outperforms E-PRV-BO significantly, even when the Δt^j_{max} parameter of E-PRV-BO is chosen to maximize η via exhaustive search.[14] We see that MSA also outperforms the upper bound to E-PRV-BO obtained under perfect forecasts, albeit by a smaller margin; however, the performance gap widens significantly beyond 5250 devices.

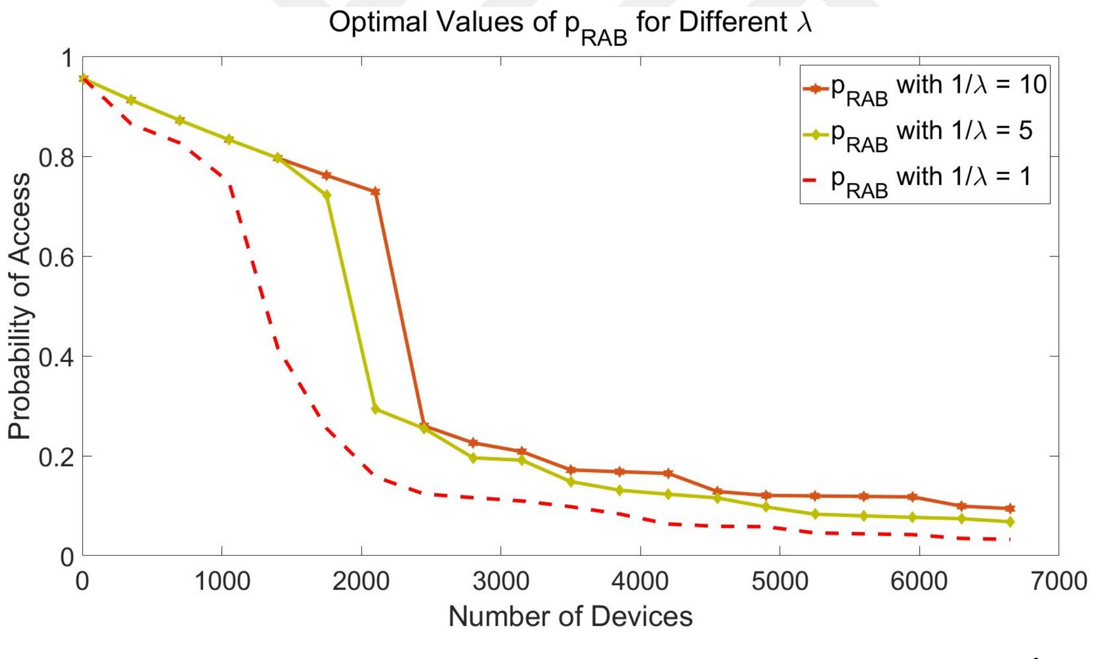

Figure 4.11. The values of p_{RAB} that maximize the throughput η for different λ's

We now present the results on energy consumption for the same simulation as for

[14]We maximized η over $\kappa > 0$ where $\Delta t^j_{max} = \kappa \Delta_j \ \forall j \in \mathscr{J}$ and found that the optimal κ is 1 in this simulation setting. However, note in Fig. 4.10 that $\Delta t^j_{max} = \infty$ produces an η that is very close to the one for $\kappa = 1$.

Fig. 4.10. We model only the transmit energy consumption[15] and assume that the uplink transmission of data in each slot consumes 1 unit of energy[16]. We let $\mathscr{E}$ denote the total transmit energy consumption of all devices divided by the total number of bits in successfully delivered bursts. In Fig. 4.12, we see that $\mathscr{E}$ for RAB is at least 2 to 4 times that of MSA when $N > 3000$, while that of PAL is higher than but comparable to that of MSA. The reason for the relatively flat behavior of $\mathscr{E}$ for MSA, PAL and E-PRV-BO as a function of N is that they schedule bits free of collisions based on forecasts. In Fig. 4.13, in which we have zoomed into the lower part of Fig. 4.12, we see that $\mathscr{E}$ of E-PRV-BO is larger than that of MSA for all N by a factor that ranges from 1.24 to 1.52.

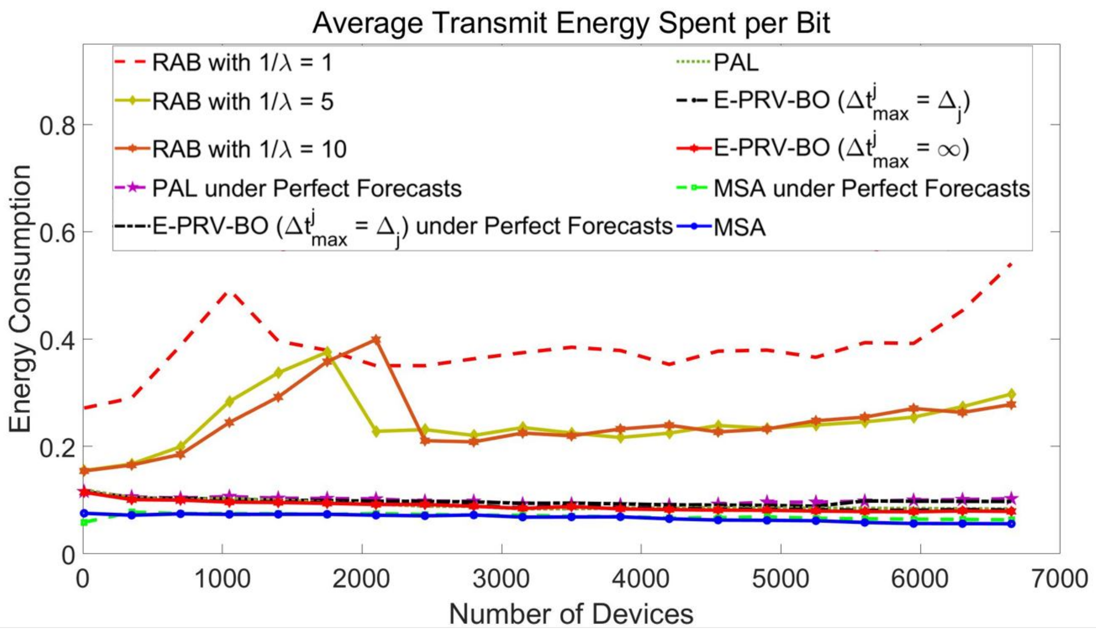

Figure 4.12. Total transmit energy consumption of all devices divided by the total number of bits in successfully delivered bursts

In order to understand the source of the gradual decrease of the throughput η of MSA as a function of N in Fig. 4.10, we graph in Fig. 4.14 the throughput of MSA for four networks each of which is made up of only a single device class. The results show that η remains almost constant in N for FBP. For VBP and VBA, η decreases for more than 2800 devices and 4400 devices, respectively. Our examination has revealed that this is not due to the underestimation of device traffic in forecasting but rather the insufficiency of the MAC-layer system capacity to deliver the traffic. For FBA, η decreases much more sharply in N. In this case, our examination has revealed that because the forecaster

[15]For many IoT devices, transmit energy is expected to dominate over idle energy consumption.

[16]This is a coarse approximation that does not take into account the differences in path loss as well as the data rate from each device to G.

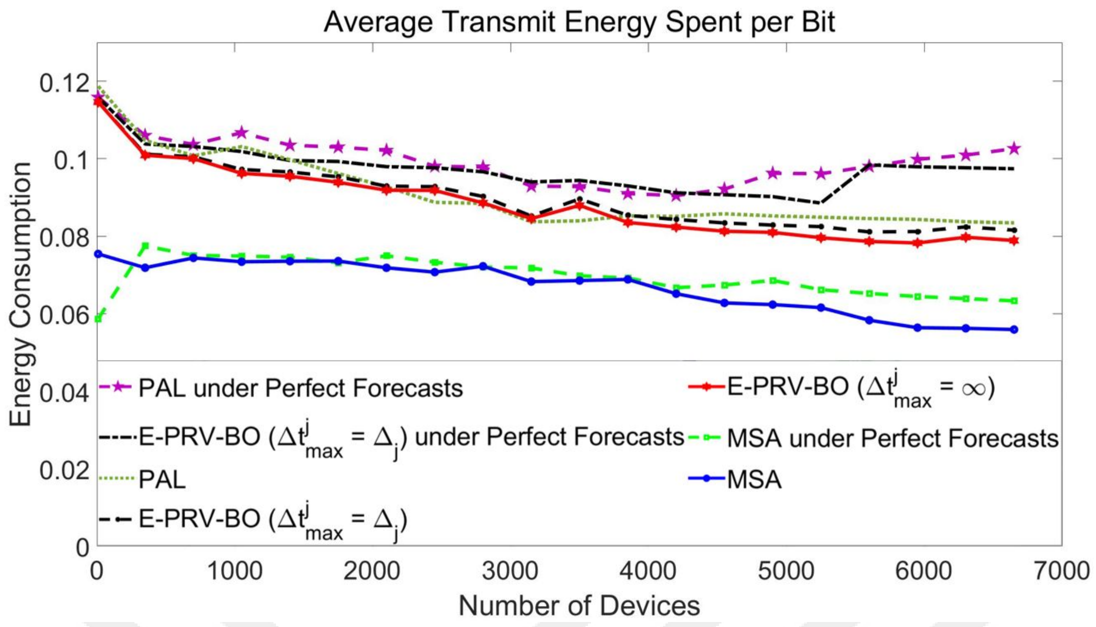

Figure 4.13. Total transmit energy consumption of only the predictive protocols, obtained by zooming into the lower part of Fig. 4.12

makes a binary decision between the two possible values of $X_i[k]$ for device i in slot k, underestimation of $X_i[k]$ typically results in a large throughput penalty since 0 slots are allocated for that burst in advance.

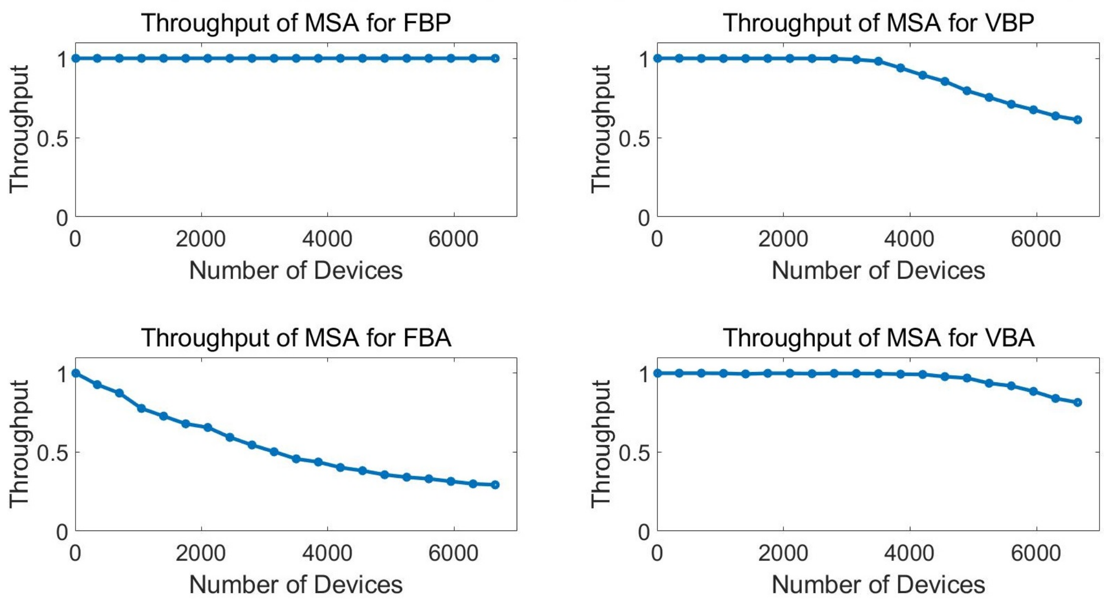

Figure 4.14. Throughput of the MSA for four networks, each of which is made up of devices from only one of the device classes FBP, VBP, FBA and VBA

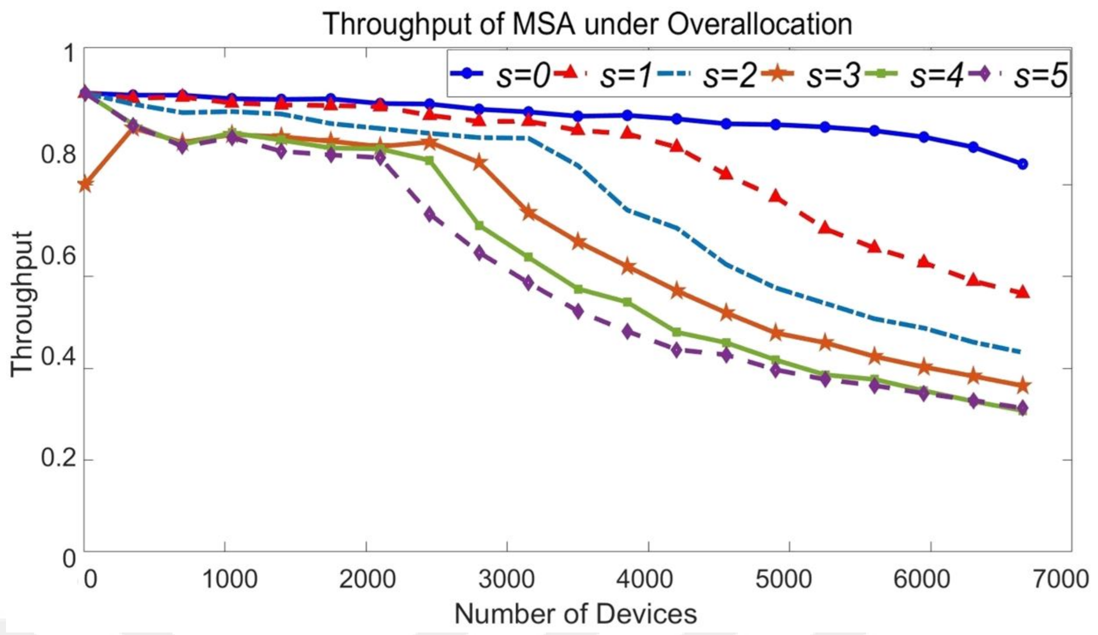

Figure 4.15. Throughput under overallocation of processing time slots in MSA

Fig. 4.15 shows η of MSA for the same set-up as in Fig. 4.10 except that we deliberately allocate s extra slots for each burst. The results show that such overallocation decreases η monotonically as a function of s across all N. This implies that accurate forecasts of the processing times play a significant role in the high throughput performance of MSA.

The extra control overhead incurred by MSA is the size of the compressed representation of each device i's generated traffic pattern since i's last transmission to G, summed over all devices. Each device encodes this traffic pattern by transmitting the slot numbers that have a positive number of generated bits as well as the number of generated bits in each such slot. (The slots are numbered in multiples of T_i for device i, taking slot 0 as the first slot whose generation pattern has not yet been reported to G.) We define the percentage control overhead of MSA as the total number of extra control bits used by all devices in this encoding over the total number of control and data bits sent by all devices. In Fig. 4.16, for the same set-up as for Fig. 4.10, we see that the percentage overhead grows slowly in N and remains less than 1.5% for up to 6650 devices.

We have used MATLAB's Artificial Neural Network (ANN) toolbox for implementing MLP and IBM ILOG CPLEX Optimization Studio 12.8 for scheduling. All simulations were run on a 4.00 GHz AMD FX(tm)-8350 8-Core Processor with 32 GB of RAM in the absence of any other concurrent compute-intensive processes. For the same set-up as for Fig. 4.10, Fig. 4.17 displays the growth of the computation time in N for only forecasting and only scheduling (excluding all other components of system simulation)

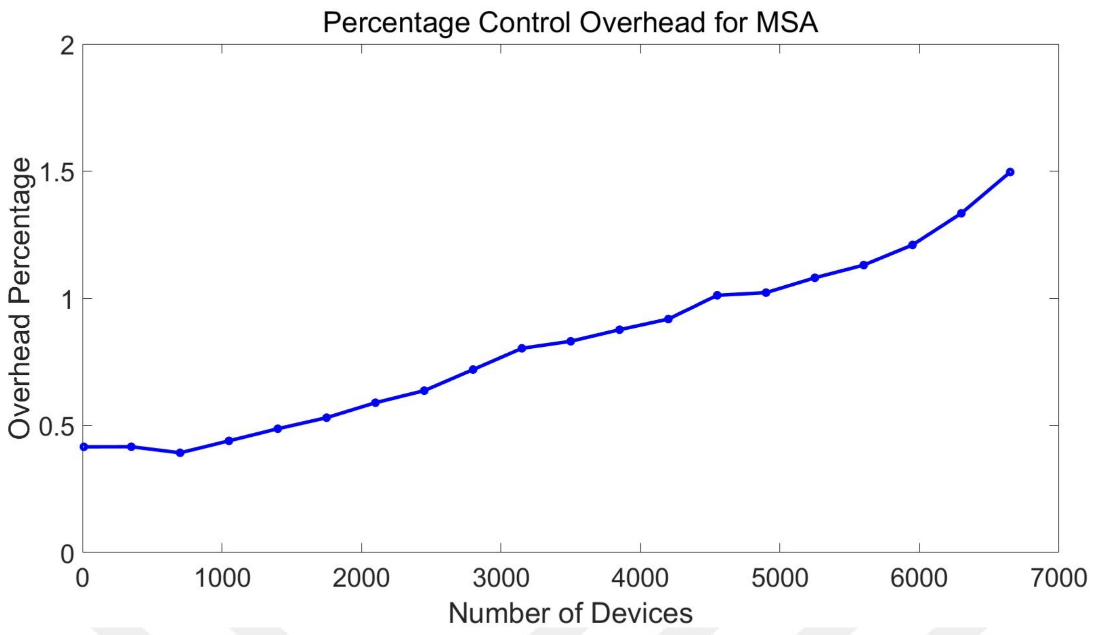

Figure 4.16. Percentage control protocol overhead of the MSA under the network simulation set-up of Fig. 4.10

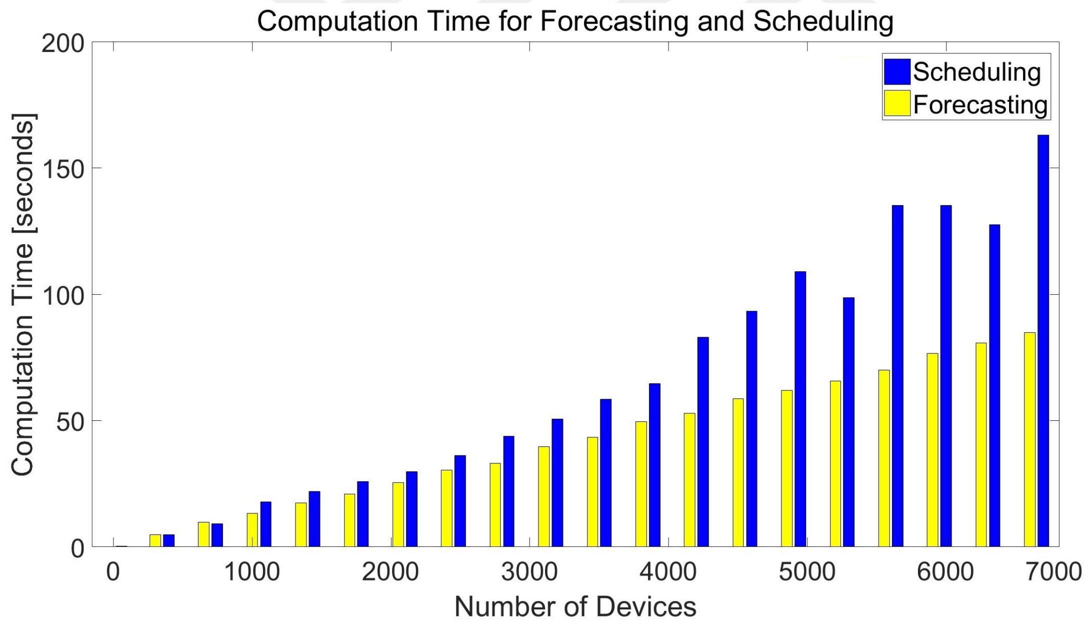

Figure 4.17. Computation time of the forecasting and the scheduling operations of MSA under the network simulation set-up of Fig. 4.10

as an indicator of the scalability of these operations if they were implemented in an actual Gateway. We see that the computation time for forecasting grows almost linearly in N, whereas that for scheduling grows superlinearly in N in most instances. The fact that the total computation time of these two operations is approximately 250 seconds for 6650 devices (compared with $T_{sch}^{(0)} = 900$ seconds) implies that the MSA can be run in real-time for a set of devices with T_{min} on the order of minutes, as used in this simulation.

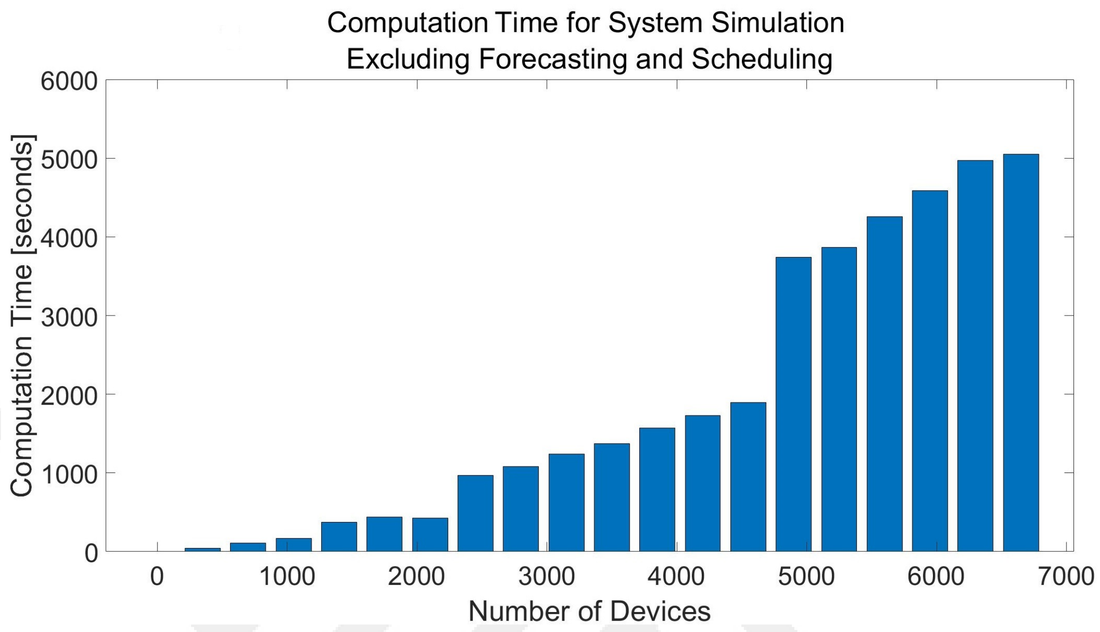

Figure 4.18. Computation time for system simulation *excluding* the forecasting and the scheduling operations, under the network simulation set-up of Fig. 4.10

Fig. 4.18 displays the total system simulation time *excluding* forecasting and scheduling for the same scenario. This system simulation time is dominated, in our current implementation, by the I/O operations performed by IBM ILOG. This simulation time, which is spent on simulating the system for the entire set of nodes, would not be incurred by an actual Gateway that runs the MSA.

4.4.5 Performance Comparison under the IEEE 802.11ah Standard

In this section, we compare the performance of the MSA against those of the other schemes under a specific scenario that utilizes a selection of data rates that are available under the IEEE 802.11ah standard ("IEEE Standard for Information technology–Telecommunications and information exchange between systems - Local and metropolitan area networks–Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub 1 GHz

License Exempt Operation", 2017). For the set of IoT devices in our simulations, this scenario under the IEEE 802.11ah standard corresponds to the case with a relatively light traffic load with respect to the total system capacity.

In our simulation, out of the possible channel bandwidths for IEEE 802.11ah, we use a single 2MHz channel for all devices, each of which utilizes the channel with a possibly different modulation scheme. The guard interval duration is 8 μs. The devices, which appear in Section 4.4.3, are assigned to the modulation schemes, code rates and the corresponding data rates (Sun, Choi, & Choi, 2013) under the IEEE 802.11ah standard in the same order as they appear in that section as follows: (1) QPSK with code rate 1/2 (1.3 Mbps), (2) 16-QAM with 1/2 (2.6 Mbps), (3) QPSK with 3/4 (1.95 Mbps), (4) 16-QAM with 3/4 (3.9 Mbps), (5) 64-QAM with 2/3 (5.2 Mbps), and (6) 64-QAM with 5/6 (6.5 Mbps). The MAC-layer slot capacities that correspond to this sequence of data rates are calculated by multiplying the data rate by $\tau_{MAC} = 0.1$ second[17]; thus, $C^{(1)} = 130$, $C^{(2)} = 260$, $C^{(3)} = 195$, $C^{(4)} = 390$, $C^{(5)} = 520$, and $C^{(6)} = 650$ kbits per slot.[18] Finally, in presenting the comparison with RAB, we find the optimal value of the backoff parameter $1/\lambda$ of RAB, as before, via exhaustive search for this network simulation set-up.

In Fig. 4.19, we compare the throughput of MSA against those of RAB, PAL, and E-PRV-BO. We also display the upper bounds for MSA, PAL and E-PRV-BO under perfect forecasts. First, we see that both the throughput of the MSA and that of the MSA under perfect forecasts are close to 1 for the entire range of the number of devices up to 6650. Second, we see that the throughput of PAL under perfect forecasts is close to that of the MSA; however, the throughput of PAL (under our MLP-based forecasting) gradually decreases to 0.828 for 6650 devices because, unlike MSA, PAL is not able to balance the load over the scheduling window due to its greedy structure. The effects of this fact become pronounced when the MLP-based forecasting overallocates the number of slots, which causes the performance of PAL to fall as the number of devices increases. Third, the throughput of E-PRV-BO under perfect forecasts falls sharply beyond 4900 devices, and that of E-PRV-BO begins to decrease relatively fast beyond 2800 devices. Fourth, we see that RAB significantly underperforms all of the predictive protocols.

In Fig. 4.20, we present the results on the transmit energy consumption per successfully

[17]We note that we do *not* implement the MAC layer of IEEE 802.11ah, since our work presents the novel MAC-layer scheme that replaces the MAC layer of IEEE 802.11ah. That is, we utilize only the Physical Layer constraints imposed by the IEEE 802.11ah in this section.

[18]We use the superscript for the sequence of capacities in order to distinguish these from C_i, which denotes the MAC-layer capacity for each device i. The C_i's take values from the above sequence of capacities that are specified using the superscript.

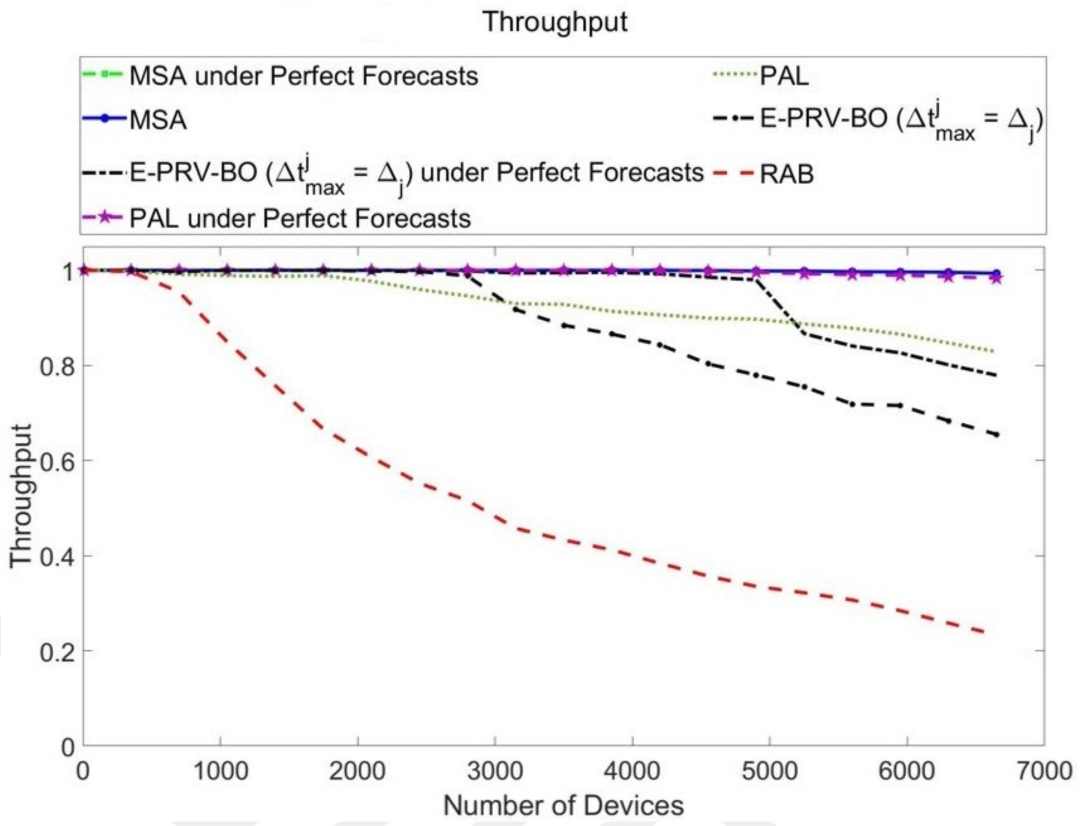

Figure 4.19. Throughput (as defined in Section 1.4) of the MSA compared with those of RAB, PAL, E-PRV-BO, and their respective upper bounds under perfect forecasts for the scenario under the IEEE 802.11ah standard

transmitted bit, denoted by $\mathscr{E}$, for MSA, PAL, E-PRV-BO, RAB, as well as those of MSA, PAL and E-PRV-BO under perfect forecasts. We see that the energy consumption of RAB is 2.3 to 16 times higher than that of MSA. Furthermore, while $\mathscr{E}$ of RAB increases sharply with N beyond 6000 devices, those of the predictive protocols remain almost constant in N. In Fig. 4.21, in which we have zoomed into the lower part of Fig. 4.20 for only the predictive protocols, we see that the values of $\mathscr{E}$ for MSA and that of MSA under perfect forecasts are equal and are approximately 85% of the values of $\mathscr{E}$ for E-PRV-BO and for E-PRV-BO under perfect forecasts. Furthermore, MSA outperforms PAL under perfect forecasts in the $\mathscr{E}$ metric for all N up to 6650 and outperforms PAL in the same metric for all $N < 5300$.

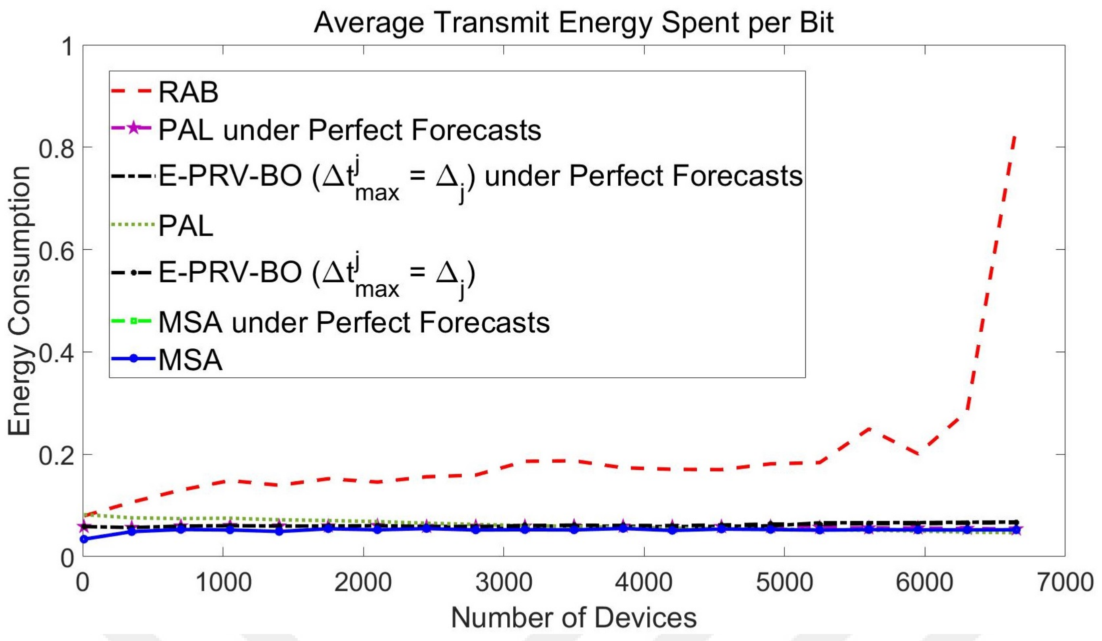

Figure 4.20. Total transmit energy consumption of all devices divided by the total number of bits in successfully delivered bursts for the scenario under the IEEE 802.11ah standard.

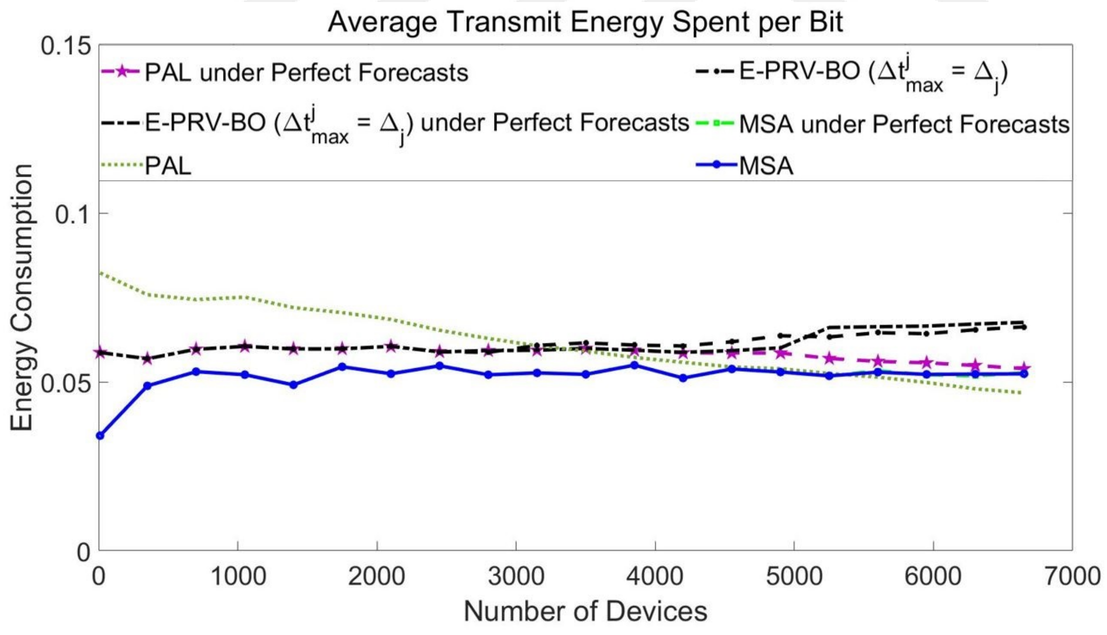

Figure 4.21. Zoom-in to the lower part of Fig. 4.20 for only the predictive protocols

4.5 Discussion on the Practical Use of the JFS System

In this section, we discuss how our joint forecasting-scheduling framework can be extended to scenarios in which IoT devices join or leave the network.

Initially, when a device i is introduced into $\mathscr{C}_G$, there is a training period by the end of which the trainable parameters of the forecaster of i at G have converged. We envision that the entire communication system will be divided into two subsystems: The first subsystem, which runs traditional random access and uplink data channels, handles the traffic of IoT devices whose forecasters are still in training. The second subsystem, which is based on joint forecasting-scheduling, handles the communication of all IoT devices whose forecasters have converged.[19] Once the forecaster for a device has converged, the device is transferred from the first to the second subsystem. In this work, we have focused on the steady-state operation of the second subsystem; that is, we have assumed that the training period of each device in $\mathscr{N}$ has already been completed.

After device i has been transferred to the second (i.e. the joint forecasting-scheduling) subsystem, G remains associated with that device at all times unless the device is taken off the network. Hence, a newly generated burst of such a device will *not* cause the device to attempt to re-form association with G. Thus, each device whose forecaster has converged no longer contends for the channel, as would be the case in reactive protocols; instead, the slots for its next forecast burst are communicated to the device by G (Section 4.2).

Finally, a signaling scheme typically exists that tracks whether each device is still on the network. Based on this signaling scheme, the forecaster for that device may be disabled once the device has been marked as off-the-network. This will prevent the allocation of any future MAC-layer slots to that device that might otherwise be allocated due to the forecasts that are based on the past traffic generation pattern of the device.

4.6 Summary

In this chapter, we developed a novel algorithm, called the Multi-Scale Algorithm, that represents the traffic generation patterns of IoT devices at multiple time scales and schedules their uplink traffic to an IoT gateway in advance based on a combination of load balancing across time blocks and scheduling at the MAC-layer slot resolution.

[19]The training of the forecaster may continue incrementally even after the forecaster has converged, as the device continues to generate new traffic.

Our main finding was that when the traffic generation pattern of an IoT device is summed across increasingly longer time blocks, thus leading to successively coarser representations of the traffic generation pattern, the future traffic can be predicted accurately over a longer time window. Based on this finding, the MSA begins by distributing the forecast traffic load of IoT devices at the coarsest level. The algorithm successively balances the traffic load at increasingly finer resolutions until at the lowest level, the algorithm performs MAC-layer scheduling of the forecast traffic. The main advantage of MSA is that it leads to manageable time and space complexity and allows a joint forecasting-scheduling system to be implemented over a much longer scheduling window that could be achieved otherwise.

In this chapter, we also compared the performance of MSA with those of reactive, joint access-scheduling schemes and showed that MSA achieves not only a much higher network throughput but also a much lower transmit energy consumption with respect to these schemes. In addition, we compared the performance of MSA against our proactive heuristic, PAL as well as E-PRV-BO. We found that MSA outperformed both of these proactive schemes. The development of the MSA for the joint forecasting-scheduling of MAC-layer IoT traffic has the potential to open the way to the solution of the Massive Access Problem in an architecture that uses a Gateway dedicated to IoT devices.

Chapter 5
CONCLUSIONS

In this thesis, in order to solve the Massive Access Problem, we have proposed Multi-Scale Joint Forecasting-Scheduling, which is a cross-layer proactive resource allocation scheme that combines forecasting of individual IoT device traffic and scheduling. This algorithm allocates the time-slot resources at the MAC layer for each of the IoT devices that are in the coverage area of an IoT gateway, denoted by G. In this system, first, G forecasts the traffic generation pattern of each IoT device over a scheduling window. Then, it aims to schedule the devices' uplink transmissions free of any collisions over this window based on its forecasts. In this thesis, first, in Chapter 2, we examined the forecasting of the IoT traffic generation pattern at the MAC layer and compared the performance of LSTM, MLP, ARIMA and the 1D CNN models. Second, in Chapter 3, we proposed the Joint Forecasting-Scheduling system and demonstrated its performance for distinct forecasting schemes as well as the PAL heuristic, which was developed as the scheduling algorithm. Finally, in Chapter 4, we proposed the Multi-Scale Algorithm (MSA), which extends the duration of the scheduling window and improves the performance of the JFS system by using representations of the traffic generation patterns at multiple time scales.

In Chapter 2, we showed that in the sMAPE metric, the traffic generation patterns in the VBP and the FBA classes appear far more predictable than those in the VBA class using the best forecasting model in each case. Furthermore, we have arrived at the following conclusions: (1) For the VBP class, the LSTM model outperforms all of the other models. (2) For the FBA class, the ARIMA model underperforms all of the other models, and the performance of LSTM, 1D CNN and MLP are comparable. (3) For the VBA class, all of the four models perform insufficiently for K-step ahead prediction for $K > 5$. For the majority of the device classes, with regard to training time, LSTM has the highest, and the ARIMA has the lowest training time out of the four forecasting models. When the forecasting error and the training time are examined jointly, we found that (1) for the VBP class, LSTM and ARIMA are comparable; (2) for the FBA class, MLP outperforms all of the other models; and (3) for the VBA class, ARIMA outperforms all of the other models.

In Chapter 3, we proposed our novel JFS system, and we examined the performance of the JFS system under perfect forecasting and under each of the LSTM, MLP and ARIMA forecasting models. In this chapter, we also proposed the heuristic scheduling algorithm PAL, which allocates the MAC-layer resources to IoT devices in the coverage area of G. We empirically showed that the time complexity of PAL is approximately linear in the number of devices. Among all of the forecasting schemes, MLP provides a reasonable trade-off between uplink cross-layer throughput and execution time. When all of the FBP, VBP, FBA and VBA classes are represented in equal proportions in a network, the throughput performance loss of MLP with respect to that of perfect forecasts is minimal. However, MLP underperforms significantly in network throughput when all of the devices in the network fall in the FBA class. In this case, a combination of LSTM and ARIMA outperforms MLP. Hence, our key conclusion is that the choice of the forecasting scheme depends heavily on the proportions with which distinct device classes are represented in the network.

In Chapter 4, we proposed a novel algorithm, namely MSA, for joint forecasting-scheduling of uplink MAC-layer IoT traffic. The MSA operates at multiple time scales that are determined by the delay constraints of IoT applications as well as the minimum traffic generation periods of IoT devices. In this chapter, we implemented Multi-Scale Forecasting (MSF) based on the MLP neural network because we have showed that MLP has the best trade-off between the throughput performance and the training time. In our design, MSA forecasts the upcoming traffic of IoT devices using an MLP architecture and preallocates the uplink wireless channel based on these forecasts. We showed that an MLP-based architecture for forecasting, combined with exact scheduling solutions, achieves high network throughput while satisfying the diverse delay deadlines of the IoT devices. Via our simulations, we demonstrated that the multi-scale nature of our algorithm ensures scalable time and space complexity to support up to 6650 IoT devices. Thus, such performance can be obtained at practically feasible time and space computational complexity such that it can be implemented at an IoT Gateway.

We compared the throughput and energy consumption of MSA with those of three protocols: (1) RAB, which is a reactive, joint access-scheduling protocol that we have designed that combines features of ACB (Ghavimi & Chen, 2015) and reservation-based scheduling (Yu et al., 2013). (2) PAL, which is given in Section 3.2.2, which is a proactive protocol that we have designed that utilizes non-preemptive priority scheduling such that the priority across devices is determined based on the time-averaged forecast remaining traffic load until the delay deadline. (3) E-PRV-BO protocol, which is a proactive protocol that enhances the PRV-PO protocol in (Hammad et al., 2017). We showed that MSA significantly outperforms all of these three protocols beyond 3000 devices. Furthermore, we showed that the percentage control overhead of MSA remains less than 1.5% for up to 6650 devices. Finally, we presented a comparison of the performance of MSA against those of the above three protocols under the IEEE 802.11ah standard. Our results pave the way to building scalable joint forecasting-scheduling engines to handle a massive number of IoT devices at IoT Gateways. Furthermore, we presented the performance of the MSA for each of the pure device classes FBP, VBP, FBA and VBA, and showed that the throughput for the MSA decreases significantly for the FBA, where this result is very similar with JFS system in Chapter 3.

In conclusion, our results suggest a joint forecasting-scheduling architecture to be implemented at an IoT Gateway, in which the Gateway predicts the future traffic of devices in distinct device classes and schedules their future traffic jointly based on these forecasts. We showed that the Multi-Scale Joint Forecasting-Scheduling achieves a much higher network performance than reactive schemes.

5.1 Contributions of the Thesis

The first contribution of this thesis is the development of a novel algorithm that operates at multiple time scales to perform joint forecasting and scheduling of IoT devices at the MAC layer. The main advantage of such a methodology is three-fold: (1) The amount of control signaling overhead can be kept to a minimum. (2) The delays that occur in traditional protocols due to contention, collisions and handshaking can be avoided. (3) If the forecasts are accurate, high network performance can be achieved via reservation-based scheduling.

The second contribution of this thesis is the design of a JFS system, to be implemented at an IoT Gateway, that forecasts the future traffic generation pattern of each IoT device and schedules the traffic of these devices in advance without any collisions over a scheduling window. We showed that not only does such a system have the advantage of potentially achieving a high throughput compared with reactive solutions to the Massive Access Problem, but it also minimizes the average transmit energy spent by each IoT device and thus prolongs the lifetime of battery-limited IoT devices.

The third contribution of this thesis is that in Section 2.3.3, we showed that the traffic generation patterns of M2M communication are far more predictable than has been assumed so far in the past work on the Massive Access Problem. While the random arrival model may be well-suited to H2M or H2H traffic, it is not necessarily suitable for M2M traffic since both the amount and the timing of machine-generated traffic can be highly predictable. In this thesis, we divided the IoT devices into four classes based on their traffic generation characteristics: (1) FBP, (2) VBP, (3) FBA, and (4) VBA. If an IoT device generates a constant number of bits over time in each generation instance, we categorize this device as “Fixed-Bit”; otherwise, we say that the device is “Variable-Bit”. Furthermore, if the device generates traffic at regularly spaced time instances, then we call the device “Periodic”; otherwise, we say that the device is “Aperiodic”.

Predictability of M2M traffic has far-reaching implications for the design of practical access schemes. For example, if all IoT traffic could be predicted accurately, then a base station or IoT Gateway could utilize a joint forecasting-scheduling architecture in which it predicts the future traffic of all IoT devices over a scheduling window and would schedule all of the traffic collision-free in advance. Thus determining to what extent *individual* IoT device traffic generation patterns are predictable is the first step towards understanding whether *proactive* solutions to the Massive Access Problem are viable.

The fourth contribution of this thesis is a comparison of the performance of JFS with respect to the forecasting scheme employed. In particular, we evaluate the network performance of ARIMA against the machine-learning based MLP and LSTM models when each of these models is used for forecasting the future traffic generation patterns of IoT devices. In this work, we measure network performance via the uplink cross-layer throughput in a full JFS system. This provides a much more accurate measure of performance than traditional performance metrics for forecasting (such as Mean Square Error and symmetric Mean Absolute Percentage Error) that do not directly quantify the impact of forecasting on scheduling. Furthermore, via our simulations, we estimate the computation time that would be required to implement JFS in an actual system that operates at an IoT Gateway.

5.2 Future Work

The development of the Multi-Scale Algorithm for the joint forecasting-scheduling of MAC-layer IoT traffic has the potential to open the way to the solution of the Massive Access Problem in an architecture that uses a Gateway dedicated to IoT devices.

In our future work, first, we shall develop scheduling heuristics as well as fast forecasting schemes in order to reduce the total computation time at the IoT gateway. Second, we shall generalize the MSA from the single channel case treated in this thesis to the case of multiple channels. Third, we shall improve the forecasting performance of the IoT device traffic generation pattern by improving both the forecasting model and the feature selection mechanism.

REFERENCES

Aijaz, A., & Aghvami, A. H. (2015). Cognitive machine-to-machine communications for Internet-of-Things: A protocol stack perspective. *IEEE Internet of Things Journal*, *2*(2), 103–112.

Aijaz, A., Ping, S., Akhavan, M. R., & Aghvami, A.-H. (2014). CRB-MAC: A receiver-based MAC protocol for cognitive radio equipped smart grid sensor networks. *IEEE Sensors Journal*, *14*(12), 4325–4333.

Alavikia, Z., & Ghasemi, A. (2018). Collision-aware resource access scheme for LTE-based machine-to-machine communications. *IEEE Transactions on Vehicular Technology*, *67*(5), 4683–4688.

Alpaydin, E. (2020). *Introduction to machine learning*. MIT press.

Alpern, N. (2009). *Eleventh hour network+: Exam n10-004 study guide*. Syngress.

Asghari, P., Rahmani, A. M., & Javadi, H. H. S. (2019). Internet of Things applications: A systematic review. *Computer Networks*, *148*, 241–261.

Bello, O., & Zeadally, S. (2019). Toward efficient smartification of the Internet of Things (IoT) services. *Future Generation Computer Systems*, *92*, 663–673.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Boutaba, R., Salahuddin, M. A., Limam, N., Ayoubi, S., Shahriar, N., Estrada-Solano, F., & Caicedo, O. M. (2018). A comprehensive survey on machine learning for networking: evolution, applications and research opportunities. *Journal of Internet Services and Applications*, *9*(1), 16.

Bouzouita, M., Hadjadj-Aoul, Y., Zangar, N., Tabbane, S., & Viho, C. (2015). A random access model for M2M communications in LTE-advanced mobile networks. In *Modeling and simulation of computer networks and systems* (pp. 577–599). Elsevier.

Braden, R. (1989). *Rfc1123: Requirements for internet hosts-application and support.* RFC Editor.

Chen, M., Wan, J., & Li, F. (2012). Machine-to-machine communications: Architectures, standards and applications. *Ksii transactions on internet & information* *systems*, *6*(2).

Chinchali, S., Hu, P., Chu, T., Sharma, M., Bansal, M., Misra, R., … Katti, S. (2018). Cellular network traffic scheduling with deep reinforcement learning. In *Thirty-second aaai conference on artificial intelligence.*

Cisco. (Mar. 2020). *Cisco Annual Internet Report (2018–2023).* Retrieved from `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`

Condoluci, M., Dohler, M., & Araniti, G. (2017). Machine-type communications over 5G systems: Challenges and research trends for supporting industrial CPS applications. In *Cyber-physical systems* (pp. 75–89). Elsevier.

De Vito, S., Massera, E., Piga, M., Martinotto, L., & Di Francia, G. (2008). On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical*, *129*(2), 750–757.

Durisi, G., Koch, T., & Popovski, P. (2016). Toward massive, ultrareliable, and low-latency wireless communication with short packets. *Proceedings of the IEEE*, *104*(9), 1711–1726.

Edalat, Y., Ahn, J.-S., & Obraczka, K. (2016). Smart experts for network state estimation. *IEEE Transactions on Network and Service Management*, *13*(3), 622–635.

Ericsson. (Nov. 2019). *Ericsson Mobility Report*. Retrieved from `https://www.ericsson.com/en/mobility-report`

Fortino, G., Russo, W., Savaglio, C., Viroli, M., & Zhou, M. (2017). Modeling opportunistic IoT services in open IoT ecosystems. In *Woa* (pp. 90–95).

Ghavimi, F., & Chen, H.-H. (2015). M2M communications in 3GPP LTE/LTE-A networks: Architectures, service requirements, challenges, and applications. *IEEE Communications Surveys & Tutorials*, *17*(2), 525–549.

Glaros, C., & Fotiadis, D. I. (2005). Wearable devices in healthcare. In *Intelligent paradigms for healthcare enterprises* (pp. 237–264). Springer.

Glitho, R. H. (2011). Application architectures for machine to machine communications: Research agenda vs. state-of-the art. In *7th international conference on broadband communications and biomedical applications* (pp. 1–5).

Grigonis, R. (2000). *Computer telephony encyclopedia.* CRC Press.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation* *computer systems*, *29*(7), 1645–1660.

Hammad, K., Moubayed, A., Primak, S. L., & Shami, A. (2017). QoS-aware energy and jitter-efficient downlink predictive scheduler for heterogeneous traffic LTE networks. *IEEE Transactions on Mobile Computing*, *17*(6), 1411–1428.

Hasan, M., Hossain, E., & Niyato, D. (2013). Random access for machine-to-machine communication in LTE-advanced networks: issues and approaches. *IEEE communications Magazine*, *51*(6), 86–93.

Haykin, S. (2010). *Neural networks and learning machines, 3/e*. Pearson Education India.

Hinton, G. E., Sejnowski, T. J., Poggio, T. A., et al. (1999). *Unsupervised learning: foundations of neural computation*. MIT press.

Ict, M. (2013). 317669-metis/d1. scenarios requirements and KPIs for 5G mobile and wireless system.

IEEE standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications amendment 2: Sub 1 GHz license exempt operation. (2017, May). *IEEE Std 802.11ah-2016 (Amendment to IEEE Std 802.11-2016, as amended by IEEE Std 802.11ai-2016)*, 1-594. doi: 10.1109/IEEESTD.2017.7920364

Igarashi, Y., Ueno, M., & Fujisaki, T. (2012). Proposed node and network models for an M2M internet. In *2012 world telecommunications congress* (pp. 1–6).

Jin, H., Toor, W. T., Jung, B. C., & Seo, J.-B. (2017). Recursive pseudo-Bayesian access class barring for M2M communications in LTE systems. *IEEE Transactions on Vehicular Technology*, *66*(9), 8595–8599.

Kuhlins, C., Rathonyi, B., Zaidi, A., & Hogan, M. (Jan. 2020). White paper: Cellular networks for massive IoT. Retrieved from `https://www.ericsson.com/en/reports-and-papers/white-papers/cellular-networks-for-massive-iot--enabling-low-power-wide-area-applications`

Lawton, G. (2004). Machine-to-machine technology gears up for growth. *Computer*, *37*(9), 12–15.

Leyva-Mayorga, I., Tello-Oquendo, L., Pla, V., Martinez-Bauset, J., & Casares-Giner, V. (2016). Performance analysis of access class barring for handling massive M2M traffic in LTE-A networks. In *2016 ieee international conference on communications (icc)* (pp. 1–6).

Liang, L., Xu, L., Cao, B., & Jia, Y. (2018). A cluster-based congestion-mitigating access scheme for massive M2M communications in Internet of Things. *IEEE Internet of Things Journal*, *5*(3), 2200–2211.

Lien, S.-Y., Liau, T.-H., Kao, C.-Y., & Chen, K.-C. (2012). Cooperative access class barring for machine-to-machine communications. *IEEE Transactions on Wireless Communications*, *11*(1), 27–32.

Lin, T.-M., Lee, C.-H., Cheng, J.-P., & Chen, W.-T. (2014). PRADA: Prioritized random access with dynamic access barring for MTC in 3GPP LTE-A networks. *IEEE Transactions on Vehicular Technology*, *63*(5), 2467–2472.

Liu, J., Song, L., et al. (2017). A novel congestion reduction scheme for massive machine-to-machine communication. *IEEE Access*, *5*, 18765–18777.

Liu, Y., Yuen, C., Cao, X., Hassan, N. U., & Chen, J. (2014). Design of a scalable hybrid MAC protocol for heterogeneous M2M networks. *IEEE Internet of Things Journal*, *1*(1), 99–111.

Lo, A., Law, Y. W., & Jacobsson, M. (2013). A cellular-centric service architecture for machine-to-machine (M2M) communications. *IEEE wireless communications*, *20*(5), 143–151.

Miorandi, D., Sicari, S., De Pellegrini, F., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad hoc networks*, *10*(7), 1497–1516.

Muralidharan, S., Roy, A., & Saxena, N. (2018). Mdp-iot: Mdp based interest forwarding for heterogeneous traffic in iot-ndn environment. *Future Generation Computer Systems*, *79*, 892–908.

Nakip, M., Gül, B. C., Rodoplu, V., & Güzeliş, C. (2019). Comparative study of forecasting schemes for IoT device traffic in machine-to-machine communication. In *Proceedings of the 2019 4th international conference on cloud computing and internet of things* (pp. 102–109).

Nakip, M., Rodoplu, V., Güzeliş, C., & Eliiyi, D. T. (2019). Joint forecasting-scheduling for the internet of things. In *2019 ieee global conference on internet of things (gciot)* (pp. 1–7).

Osseiran, A., Monserrat, J. F., & Marsch, P. (2016). *5G mobile and wireless communications technology*. Cambridge University Press.

Pang, Y.-C., Chao, S.-L., Lin, G.-Y., & Wei, H.-Y. (2014). Network access for M2M/H2H hybrid systems: A game theoretic approach. *IEEE Communications Letters*, *18*(5), 845–848.

Park, I., Kim, D., & Har, D. (2015). MAC achieving low latency and energy efficiency in hierarchical M2M networks with clustered nodes. *IEEE Sensors Journal*, *15*(3), 1657–1661.

Petkov, V., & Obraczka, K. (2011). The case for using traffic forecasting in schedule-based channel access. In *2011 IEEE consumer communications and networking conference (ccnc)* (pp. 208–212).

Petkov, V., & Obraczka, K. (2012). Collision-free medium access based on traffic forecasting. In *2012 IEEE international symposium on a world of wireless, mobile and multimedia networks (wowmom)* (pp. 1–9).

Potts, C. N., & Van Wassenhove, L. (1988). Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science*, *34*(7), 843–858.

Rodoplu, V., Nakıp, M., Eliiyi, D. T., & Güzelis, C. (2020). A multi-scale algorithm for joint forecasting-scheduling to solve the massive access problem of IoT. *IEEE Internet of Things Journal*, 1-1.

Russel, S., Norvig, P., et al. (2013). *Artificial intelligence: a modern approach*. Pearson Education Limited.

Samaoui, S., El Bouabidi, I., Obaidat, M. S., Zarai, F., & Mansouri, W. (2015). Wireless and mobile technologies and protocols and their performance evaluation. In *Modeling and simulation of computer networks and systems* (pp. 3–32). Elsevier.

Shahin, N., Ali, R., & Kim, Y.-T. (2018). Hybrid slotted-CSMA/CA-TDMA for efficient massive registration of IoT devices. *IEEE Access*, *6*, 18366–18382.

Shaw, K. (2017). The OSI model explained: How to understand (and remember) the 7 layer network model. *NetworkWorld [online Dec 4, 2017][retrived online 5/30/18]*.

Shirvanimoghaddam, M., Dohler, M., & Johnson, S. J. (2017). Massive non-orthogonal multiple access for cellular IoT: Potentials and limitations. *IEEE Communications Magazine*, *55*(9), 55–61.

Si, P., Yang, J., Chen, S., & Xi, H. (2015). Adaptive massive access management for QoS guarantees in M2M communications. *IEEE Transactions on Vehicular Technology*, *64*(7), 3152–3166.

Stallings, W. (2009). *Operating systems: internals and design principles*. Upper Saddle River, NJ: Pearson/Prentice Hall,.

Sun, W., Choi, M., & Choi, S. (2013). IEEE 802.11 ah: A long range 802.11 WLAN at sub 1 GHz. *Journal of ICT Standardization*, *1*(1), 83–108.

Tello-Oquendo, L., Leyva-Mayorga, I., Pla, V., Martinez-Bauset, J., Vidal, J.-R., Casares-Giner, V., & Guijarro, L. (2018). Performance analysis and optimal access class barring parameter configuration in LTE-A networks with massive M2M traffic. *IEEE Transactions on Vehicular Technology*, *67*(4), 3505–3520.

Tello-Oquendo, L., Pacheco-Paramo, D., Pla, V., & Martinez-Bauset, J. (2018). Reinforcement learning-based ACB in LTE-A networks for handling massive M2M and H2H communications. In *2018 IEEE international conference on communications (icc)* (pp. 1–7).

Terroso-Saenz, F., González-Vidal, A., Ramallo-González, A. P., & Skarmeta, A. F. (2019). An open IoT platform for the management and analysis of energy data. *Future Generation Computer Systems*, *92*, 1066–1079.

University of California Irvine Machine Learning Repository, . (2008). *Air Quality Data Set*. Retrieved from `http://archive.ics.uci.edu/ml/datasets/air+quality`

Verma, P. K., Verma, R., Prakash, A., Agrawal, A., Naik, K., Tripathi, R., … Abogharaf, A. (2016). Machine-to-machine (M2M) communications: A survey. *Journal of Network and Computer Applications*, *66*, 83–105.

Vodafone. (Apr. 2010). RACH intensity of time controlled devices. *3GPP TSG RAN WG2 #69bis, R2-102296*. Retrieved from `http://www.3gpp.org/DynaReport/TDocExMtg--R2-69b--28031.htm`

Wiering, M., & Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, *12*, 3.

Yu, X., Navaratnam, P., & Moessner, K. (2013). Resource reservation schemes for IEEE 802.11-based wireless networks: a survey. *IEEE Communications Surveys & Tutorials*, *15*(3), 1042-1061.

Zaidi, A., Athley, F., Medbo, J., Gustavsson, U., Durisi, G., & Chen, X. (2018). *5G physical layer: principles, models and technology components*. Academic Press.

Zaidi, A., Hussain, Y., Hogan, M., & Kuhlins, C. (Jan. 2019). White paper: Cellular IoT evolution for industry digitalization. Retrieved from `https://www.ericsson.com/en/reports-and-papers/white-papers/cellular-iot-evolution-for-industry-digitalization`

Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things journal*, *1*(1), 22–32.

Zanella, A., Zorzi, M., dos Santos, A. F., Popovski, P., Pratas, N., Stefanovic, C., … Norp, T. A. (2013). M2M massive wireless access: Challenges, research issues, and ways forward. In *2013 IEEE Globecom Workshops* (pp. 151–156).

Zimmermann, H. (1980). OSI reference model-the ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, *28*(4), 425–432.