Dynamic Automatic Forecaster Selection via Artificial Neural Network Based Emulation to Enable Massive Access for the Internet of Things

Mert Nakıp[*,a], Erdem Çakan[b], Volkan Rodoplu[b], Cüneyt Güzeliş[b]

[a]*Institute of Theoretical and Applied Informatics, Polish Academy of Sciences (PAN), Gliwice, Poland*
[b]*Department of Electrical and Electronics Engineering, Yaşar University, İzmir, Turkey*

Abstract

The Massive Access Problem of the Internet of Things (IoT) occurs at the uplink Medium Access Control (MAC) layer when a massive number of IoT devices seek to transfer their data to an IoT gateway. Although recently proposed predictive access solutions that schedule the uplink traffic based on forecasts of IoT device traffic achieve high network performance, these solutions depend heavily on the performance of forecasters. Hence, the design and selection of forecasting schemes are key to enabling massive access for such predictive access solutions. To this end, in this paper, first, we develop a framework that emulates the relationship between the IoT device class composition in the coverage area of an IoT gateway and the resulting network performance by virtue of an Artificial Neural Network (ANN). Second, based on this framework, we develop the Dynamic Automatic Forecaster Selection (DAFS) method, which selects the best-performing forecasting scheme for predictive access, in particular for Joint Forecasting-Scheduling (JFS), in a manner that adapts dynamically to a changing number of IoT devices in each device class in the coverage area. We evaluate the performance of DAFS via simulations and show that our method is able to achieve at least 80% of the best performance that can be attained for both throughput and energy consumption. Furthermore, we demonstrate that DAFS is robust with respect to the selection of architectural parameters and has a reasonable computation time for real-time IoT applications. These results imply that DAFS holds the potential for practical implementation at IoT gateways in order to enable massive access under a dynamically changing composition of IoT devices.

Keywords: Internet of Things (IoT), massive access, forecasting, artificial neural network (ANN), Medium Access Control (MAC) layer, predictive network, joint forecasting-scheduling

1. Introduction

The smart cities of the near future will consist of a plethora of wireless sensor devices, such as smart lampposts, smart garbage bins as well as a swarm of vehicles that carry Global Positioning System (GPS) devices for fleet management applications [1]. These devices are expected to be connected to the Internet in order to enable to a wide range of services to the inhabitants of smart cities. The Internet of Things (IoT) refers to the collection of these new devices, most of which operate without human intervention [2]. The next-generation Internet must support tens of thousands of such IoT devices in the coverage area of a single base station or IoT Gateway. The problem of providing wireless access to such a massive number of devices is referred to as the Massive Access

⋆Preprint published at JNCA with DOI: https://doi.org/10.1016/j.jnca.2022.103360

⋆⋆This work has been supported by TÜBİTAK (Scientific and Technological Research Council of Turkey) under the 1001 program grant no. 118E277.

*Corresponding author

Email addresses: mnakip@iitis.pl (Mert Nakıp*), cakanerdem5@gmail.com (Erdem Çakan), volkan.rodoplu@yasar.edu.tr (Volkan Rodoplu), cuneyt.guzelis@yasar.edu.tr (Cüneyt Güzeliş)

Problem.

The majority of the past approaches [3, 4, 5, 6, 7, 8] to the Massive Access Problem have centered on alleviating collisions on the Physical Random Access Channel (PRACH) in cellular systems [2]. The key assumption behind these past approaches is that each IoT device generates traffic that is modeled by "random arrivals". Since traffic generation is assumed to occur at random times in random amounts, almost all of the past access schemes merely react to the *current* traffic demand that occurs on the network. As such, they fall under the class of "reactive protocols". While such a model based on random arrivals may be suitable for the traditional Internet traffic that is comprised of Human-to-Human (H2H), Human-to-Machine (H2M) and Machine-to-Human (M2H) traffic classes, it is not necessarily suitable for Machine-to-Machine (M2M) traffic.

Recent works [9, 10] have demonstrated that the M2M traffic generated by individual IoT devices can be predicted by using machine learning algorithms. Furthermore, based on this predictability, a novel method called "Joint Forecasting-Scheduling (JFS)" was developed in [11]. In JFS, in contrast with reactive protocols, an IoT Gateway forecasts the future traffic of individual IoT devices in its coverage area and allocates Medium Access Control (MAC)-layer resources to these devices in advance based on traffic forecasts in a collision-free manner. In [12] and [13], it was demonstrated that JFS achieves a superior performance in network throughput as well as transmit energy consumption compared with benchmark reactive protocols.

In [9], IoT devices were classified into four classes with respect to their traffic generation patterns: A Fixed-Bit Periodic (FBP) device generates a constant number of bits at regular intervals. A Fixed-Bit Aperiodic (FBA) device generates a constant number of bits at irregular intervals. A Variable-Bit Periodic (VBP) device generates a variable number of bits at regular intervals. Finally, a Variable-Bit Aperiodic (VBA) device generates a variable number of bits at irregular intervals.

In References [11, 12, 13, 14, 15], the performance of JFS was demonstrated only for those cases in which either the percentage of IoT devices from each device class was identical or all of the devices in the network were from the same device class. However, in actual networks, the number of IoT devices from each device class typically changes dynamically especially due to the presence of mobile IoT devices (e.g. as in fleet management applications) that roam into and out of the coverage area. Hence, a novel approach is required that addresses a dynamically changing number of devices in each class in the coverage area of an IoT gateway and optimizes the performance of JFS with respect to the current composition of these device classes.

The first contribution of this paper is the design of such a method, which we call "Dynamic Automatic Forecaster Selection (DAFS)". We demonstrate that this method can be executed in real time and scales well with a growing number of IoT devices. In addition, the ability to adapt fast to dynamic network conditions by automatically switching the forecasting scheme used is a significant step towards performance provisioning for IoT in wireless networks in the near future.

The second contribution of this paper is the development of a novel methodology for DAFS. In our methodology, for the forecasting schemes, an Artificial Neural Network (ANN) emulates the relation between the vector comprised of the number of IoT devices in each device class and the resulting network performance. After the training of this ANN has been completed, the ANN outputs the forecaster that is estimated to have the best performance out of all of the forecasting schemes examined for each IoT device class composition. We quantify the performance of our DAFS method by comparing the actual network performance obtained under the forecaster selected by DAFS versus the forecaster that, in fact, produces the best network performance. In our demonstrations, on average, DAFS is able to achieve a network throughput performance and energy consumption performance that is 80% of the best performance that can be achieved. We note that although the performance of DAFS is evaluated for JFS, it can be used for the selection of the best performing forecasting scheme in any predictive solution technique for Massive Access Problem.

The rest of this paper is organized as follows: In

Section 2, we describe the relationship of this work to the existing work in this area. In Section 3, we state the assumptions that underlie our work. In Section 4, we review the basic JFS system. In Section 5, we present the design of the DAFS method. In Section 6, we evaluate the performance of DAFS. In Section 7, we present our conclusions.

2. Relationship to The State of The Art

In this section, we describe the relationship between our work and the state of the art in three categories: (1) We compare our work against the works that develop meta-MAC protocols which select, reconfigure or generate MAC protocols and do not utilize machine learning techniques. (2) We compare our work against meta-MAC protocols, which are designed by using machine learning algorithms. (3) We present the differences between our work and the past articles that focus on the massive access problem (including the works based on JFS).

First, we compare our work against those works, which develop meta-MAC protocols that do not utilize machine learning methods. Reference [16] creates a meta-MAC layer to switch automatically between the existing MAC-layer protocols (such as slotted ALOHA and Time-Division Multiple Access (TDMA)). Reference [17] develops a MAC design framework to compose a MAC protocol based on functions that are defined with respect to the requirements of the application. Reference [18] develops a hybrid protocol for which the running mode is adaptively switched between Carrier-Sense Multiple Access (CSMA) and TDMA for mobile ad hoc networks. In addition, Reference [19] presents adaptive MAC protocol which switches between CSMA and TDMA for the cognitive radio (CR) networks. Reference [20] presented a technique to identify the MAC protocol which is considered as a parameter in CR and to choose the best coexisting CR access schemes based on this identification. Reference [21] develops a MAC framework that switches between Carrier Sense Multiple Access (CSMA) and TDMA for Unmanned Aerial Vehicle (UAV) in ad hoc networks. Reference [22] switches between MAC protocols (e.g. CSMA/CA and TDMA) adaptive to the changes in the network conditions.

Reference [23] develops a Randomized Weighted Majority (RWM) meta-MAC protocol that dynamically optimizes the parameters of the MAC protocol without any advance knowledge on network conditions. Reference [24] develops a framework, which is called MultiMAC, that adaptively reconfigure MAC layer properties. In addition, Reference [25] automatically generates MAC protocols via an optimization program based on symbolic Monte Carlo simulation for multiple neighborhoods under dynamic topologies and dynamic traffic conditions. All of the references in this category aim to either generate a protocol or switch between existing protocols. In contrast, our DAFS method dynamically switches among the existing forecasting schemes (Multi-Layer Perceptron (MLP), Long-Short Term Memory (LSTM), and Autoregressive Integrated Moving Average (ARIMA)) for JFS by estimating the performance of the protocol under each forecasting scheme via machine learning.

Second, we present the comparison of our work with the works in the second category that develop meta-MAC protocols based on machine learning algorithms. Reference [26] uses machine learning methods (such as Naive Bayes and Random Forest) in order to select an adaptive MAC-layer protocol between CSMA and TDMA based on the traffic packet information (e.g. packet length and inter-arrival time). Reference [27] uses a Support Vector Machine (SVM) to switch between TDMA, CSMA with Collision Avoidance (CSMA/CA), pure ALOHA, and slotted ALOHA protocols for Cognitive Radio (CR) networks. Reference [28] presents the Organic Network Control (ONC) architecture, which is based on evolutionary algorithms and adapts the parameters of the protocol towards the changes in the network conditions.

Reference [29] develops a meta-MAC protocol that learns the expected performance of the MAC protocols (e. g. ALOHA and TDMA) in order to emulate the behavior of each protocol and switch between these protocols. Reference [30] develops a protocol that switches between CSMA/CA and TDMA via an SVM-based algorithm for a changing traffic load. In addition, for CR networks in order to

select between TDMA, CSMA/CA, pure ALOHA, and slotted ALOHA, Reference [31] uses SVM, Reference [32] uses Convolutional Neural Network (CNN) with spectrogram, and Reference [33] uses LSTM. The works in this category use machine learning-based algorithms to adaptively switch between existing MAC protocols or reconfigure the parameters of a protocol. In contrast, our DAFS method, which is based on ANN, selects the best forecaster specifically for JFS by estimating the performance of JFS under each forecaster for the dynamically changing IoT network conditions.

Third, we compare our work with the recent articles that focus on the predictive solutions for massive access problem including the ones based on JFS. While the existing reactive network protocols are breaking down most of the time including when a massive number of devices requests wireless access[34, 35, 36], the recent works [37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 11, 12, 13, 14, 15] aim to develop proactive solutions in which the network traffic is predicted to enable grant free medium access. In Reference [37], a subset of packets is dropped considering the predicted impact of each packet on the latency and the network performance. For delay-critical applications in industrial wireless networks, Reference [38] has scheduled uplink access of IoT devices predicting the activity of those. Recently, Reference [39] has recently developed a predictive network architecture in which they predict total network traffic analyzing the causal relationship between the occurrences of special events and the triggered traffic variations. Reference [40] performed diffusion analysis and developed quasi-deterministic packet transmission policy to mitigate the massive access in IoT networks.

A recent trend of research has focused on a proactive solution technique, called Fast Uplink Grant (FUG) which schedules the uplink grants for traffic packets based on the advance knowledge on the network traffic [41, 42, 43]. To develop enhanced FUG algorithms, Reference [44] modeled IoT packet tranmissions using binary Markovian process, and Reference [45] predicted IoT network traffic using SVM and LSTM. Moreover, Reference [46] used Nonlinear autoregressive exogenous (NARX) neural network to predict IoT network traffic while Reference [47] performed comparative study of machine learning models (e.g. Random Forest, Decision Tree and k-Nearest Neighbor) and deep neural network to classify the category of The Onion Router (Tor) network traffic.

Furthermore, Reference [11] proposes JFS, which performs resource allocation in advance based on the forecasting of the traffic generation pattern of individual IoT devices; scheduling is performed via a heuristic only for the single-channel case. Although we consider the JFS as one of the main use cases of the DAFS method and use it to evaluate DAFS in this paper, DAFS is a method that selects the best forecasting scheme dynamically by estimating the overall network performance under each forecaster candidate. Thus, DAFS is eligible to be used under any network paradigm (not only JFS) that utilizes forecasting. Reference [12] develops a multiscale algorithm, which allows JFS to operate over much longer scheduling windows than [11] does for a massive number of devices. Reference [13] extends JFS to the multi-channel case. Reference [14] develops an Application Specific Error Function (ASEF), which measures the effects of the forecasting error on network performance in JFS. Reference [15] develops a queueing theory based technique for a scheduling heuristic and evaluates the performance of this heuristic for JFS. Compared with all of these previous works, our DAFS method is a meta-level framework that selects the best forecasting scheme for JFS by emulating the relationship between the composition of the number of IoT devices in each class and the resulting network performance.

3. Assumptions

Throughout this work, we assume that a single IoT Gateway, denoted by G, runs JFS for all IoT devices in its coverage area, which is denoted by C_G. The set of IoT devices, which is denoted by $\mathcal{N}$, is assumed to be variable[1] in this work. Each IoT device (which will be called "device" for short) that falls in C_G at any given time is assumed to have a direct wireless link to G at that time. Furthermore, we

[1]This reflects the fact that the set of devices in the coverage area may change dynamically in an actual network.

Table 1: List of mathematical symbols in order of appearance

Symbol	Definition
G	Single IoT Gateway
C_G	Coverage area of Gateway G
N	Total number of devices in C_G
$\mathcal{N}$	Set of N IoT devices
Δ_i	Delay constraint for device i
N_{FBP}	Number of IoT devices that fall in FBP class
N_{VBP}	Number of IoT devices that fall in VBP class
N_{FBA}	Number of IoT devices that fall in FBA class
N_{VBA}	Number of IoT devices that fall in VBA class
$\mathbf{N}$	Vector of number of IoT devices in each class
η_{ARIMA}	Throughput performance of the JFS system under ARIMA forecaster
η_{LSTM}	Throughput performance of the JFS system under LSTM forecaster
η_{MLP}	Throughput performance of the JFS system under MLP forecaster
$\mathcal{E}_{\text{ARIMA}}$	Energy consumption of the JFS system under ARIMA forecaster
$\mathcal{E}_{\text{LSTM}}$	Energy consumption of the JFS system under LSTM forecaster
$\mathcal{E}_{\text{MLP}}$	Energy consumption of the JFS system under MLP forecaster
s_p^i	Time index, relative to the current time, that corresponds to the pth feature of the traffic generation pattern of device i
F_i	Set of indices of all selected features
$\{x_i[m - s_p^i]\}_{p\in\{1,\ldots,\lvert F_i\rvert\}}$	Collection of the past traffic generation pattern of device i over the selected features at current slot m
K_i	Number of forecasting steps for device i
$\{\hat{x}_i[m + k]\}_{k\in\{1,\ldots,K_i\}}$	Collection of the future traffic generation pattern over the forecasting steps at current slot m
$\mathbf{S}$	Binary schedule matrix
$\mathbf{N}^v$	Realization of $\mathbf{N}$ for sample v
η^v	Corresponding η vector that is collected via JFS system for sample v
$\tilde{\mathbf{N}}$	A 4-partition of 10
$\mathcal{X}$	Any network performance metric used in the JFS
$f_{\mathcal{X}}$	Custom error metric to measure the performance of DAFS
$\mathcal{X}_{\text{best}}$	The best network performance metric that is achieved by JFS system over all forecasting schemes
$\mathcal{X}_{\text{worst}}$	The worst network performance metric that is achieved by JFS system over all forecasting schemes
$\mathcal{X}_{\text{DAFS}}$	Network performance metric that is achieved by the JFS system under DAFS
E	Number of layers in the Network Performance Estimator of DAFS
h_e	Number of neurons at each layer $e \in \{1, \ldots, E\}$

assume that whenever a device falls in C_G, it remains associated with G (regardless of whether it generates traffic to be sent to G at that instance) until the device exits C_G.

We adopt the same assumptions on delay-constrained IoT applications as in [11] and [12]. In particular, we assume that a single IoT application is run at each device, which generates traffic to be delivered to G within a delay constraint, which is denoted by Δ_i for device i. Based on the forecast of the future traffic of each device, JFS schedules the future bursts of traffic that are generated by the devices in C_G at that time over a scheduling window. We define "throughput" as the ratio of the number of bits in successfully delivered bursts to the total number of bits of traffic offered by $\mathcal{N}$ over a scheduling window. We assume that an IoT device consumes 1 unit of energy for data transmission to an IoT gateway in a single slot.[2] We define $\mathcal{E}$ as the average transmit energy consumed across all IoT devices in the coverage area per successfully delivered bit.

Throughout this work, we shall let N_{FBP}, N_{VBP}, N_{FBA}, N_{VBA} denote the number of IoT devices, respectively, in the FBP, VBP, FBA and VBA device classes in C_G at any given time. We let the vector $\mathbf{N} \equiv [N_{\text{FBP}}, N_{\text{VBP}}, N_{\text{FBA}}, N_{\text{VBA}}]$.[3] Let N denote the total number of devices in C_G. (That is, $N = |\mathcal{N}|$.) Then, $N = N_{\text{FBP}} + N_{\text{VBP}} + N_{\text{FBA}} + N_{\text{VBA}}$. We also let η_{ARIMA}, η_{LSTM}, η_{MLP} denote the throughput performance of JFS under the ARIMA, LSTM and MLP forecasters, respectively. Furthermore, we define the vector $\eta \equiv [\eta_{\text{ARIMA}}, \eta_{\text{LSTM}}, \eta_{\text{MLP}}]$. Similarly, $\mathcal{E}_{ARIMA}$, $\mathcal{E}_{LSTM}$, $\mathcal{E}_{MLP}$ denotes the energy consumption of JFS under the ARIMA, LSTM and MLP forecasters, and $\mathcal{E} \equiv [\mathcal{E}_{\text{ARIMA}}, \mathcal{E}_{\text{LSTM}}, \mathcal{E}_{\text{MLP}}]$.

4. Review of Joint Forecasting-Scheduling (JFS)

In this section, we review JFS, which was developed in [11]. In Fig. 1, we present the structure of JFS, which consists of the Bank of Forecasters and a Scheduler. In the Bank of Forecasters, there is one forecaster for each device i, which is shown as Forecaster$_i$ in Fig. 1, where the Forecaster$_i$ is the forecaster that estimates the future traffic generation of device i.

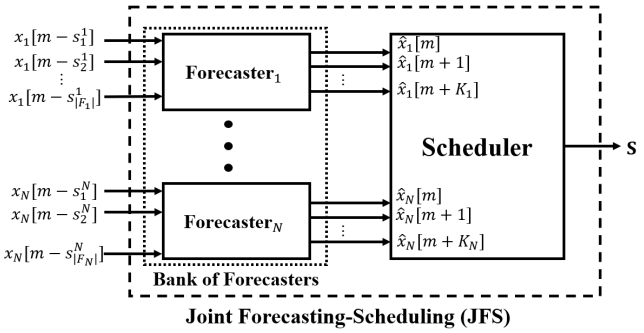


Figure 1: JFS architecture, as developed in [11]

For each device i, the input of Forecaster$_i$ is the collection of the past traffic generation pattern of that device, denoted by $\{x_i[m - s_p^i]\}_{p\in\{1,\ldots,|F_i|\}}$, where s_p^i is the time index, relative to the current time m, that corresponds to the pth feature of the traffic generation pattern of device i, and F_i is the set of indices of all selected features of device i.[4] The output of Forecaster$_i$ is the future traffic generation pattern, $\{\hat{x}_i[m + k]\}_{k\in\{1,\ldots,K_i\}}$, where K_i is the number of steps ahead for forecasting the traffic of device i.

As shown in Fig. 1, after the future traffic patterns of all of the devices in the coverage area have been forecast, those forecasts are passed to the Scheduler. The Scheduler in JFS generates a binary schedule matrix $\mathbf{S}$, whose entry (j, m) equals 1 if the MAC-layer slot m is allocated for burst j and equals 0 otherwise.

5. Design of The Dynamic, Automatic Forecaster Selection (DAFS) Method

In this section, we describe the DAFS method, which enables the automatic selection of the best-performing forecasting scheme for JFS for a

[2] This is an approximation that does not take into account the fact that each IoT device in the coverage area may see a distinct path loss to the IoT gateway.

[3] The reason that we do not use a time index t is that our DAFS method will select a forecaster based on $\mathbf{N}$ at any given time; that is, the selection of the forecaster is performed based directly on $\mathbf{N}$ in a time-invariant fashion.

[4] The selection of important features is performed in order to improve the performance of the forecaster. We use the same set of selected features as in [11].

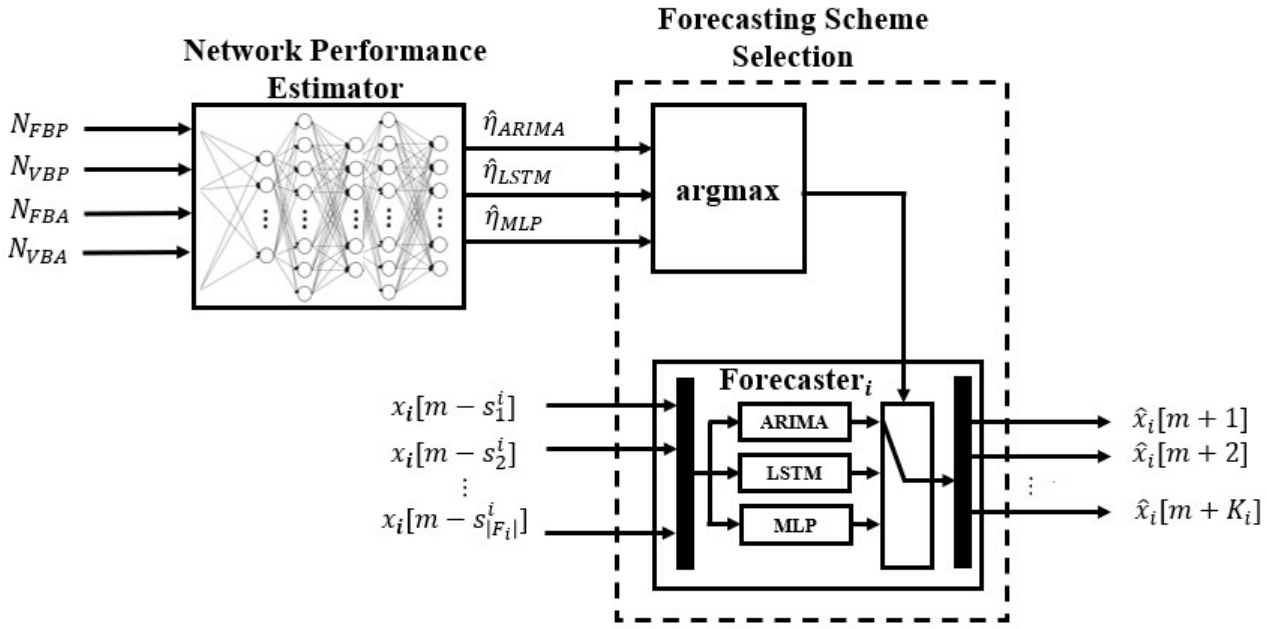


Figure 2: The architecture for the Forecaster of device *i* under DAFS for JFS has been drawn for the particular network performance metric, throughput denoted by η, as an example.

dynamically changing number of devices in each device class.

In Fig. 2, we present the architecture for our DAFS. As shown in this figure, DAFS is comprised of the Network Performance Estimator and the Forecasting Scheme Selection. The latter is comprised of the *argmax* block as well as the Selection block which selects among the ARIMA, LSTM, and MLP schemes in the forecaster (which are taken, in this paper, as examples of the forecasting schemes that can be utilized). The reason that the ARIMA, LSTM, and MLP are selected as the forecasting scheme candidates is to represent statistics-based forecasters with ARIMA, recurrent forecasters with LSTM, and feedforward multi-layer networks including deep neural networks with MLP. Also, note that in Fig. 2, in order to simplify the description of the methodology of DAFS, we show the design of DAFS for throughput; however, it might be easily used for any other network performance metric by replacing throughput with that metric in the Network Performance Estimator block.

5.1. Network Performance Estimation Based on Emulation of JFS by ANN

First, we describe the operation of the Network Performance Estimator in the design of the DAFS method in Fig. 2. The Network Performance Estimator aims to compute the value of the network performance metric. This value is subsequently used by DAFS in the selection of the best-performing forecasting scheme for the Bank of Forecasters in Fig. 1 for each composition of the number of IoT devices in each IoT device class.

The network performance metrics can be computed in a brute force manner by executing the complete JFS system under each forecasting scheme as well as the calculation of the resulting network performance. However, there are two reasons that the brute force computation cannot be used in practice for the selection of the forecasting scheme: 1) The forecasts produced by the forecasting scheme in the Bank of Forecasters are required during the execution of JFS, but the best-performing forecasting scheme is not known in advance. Thus, JFS must be executed for all available forecasting schemes (ARIMA, LSTM and MLP in this paper) in order to select the scheme that achieves the best

network performance in this case. That is, this brute force method does not allow DAFS to select the forecasting scheme before the execution of JFS for all forecasting schemes. 2) The execution time of JFS has been demonstrated to increase linearly with the number of devices in the coverage area of the IoT gateway [11]. Thus, the execution of JFS to select a forecasting scheme for the networks with a massive number of devices will require unacceptably high computation time.

5.1.1. *Emulating the Computation of Network Performance of JFS under Each Forecasting Scheme*

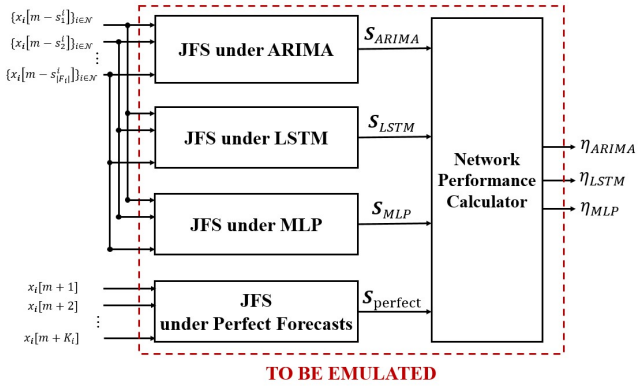


Figure 3: The system to be emulated, which computes the network performance metrics for each forecasting scheme, illustrated for the case of throughput η.

We now describe how to emulate the computation of the network performance metric of JFS, which is presented in Fig. 3. The network performance metric should be calculated based on the schedule matrix which is computed by JFS. Therefore, in order to compute this metric under each of the ARIMA, LSTM, and MLP forecasting schemes, JFS must be executed under each of these schemes as well as under perfect forecasts. However, since the execution of JFS under four different forecasting schemes is computationally intensive, we shall emulate the system which is comprised of four parallel JFS systems[5] and the Network Performance Metric Calculator.

[5]There is one JFS system under each of ARIMA, LSTM and MLP forecasting; there is a fourth JFS system under perfect forecasts.

Via the emulation of this system, the vector of network performance metrics, in this case η, will be computed. As shown in Fig. 3, during execution, the input is the collection of the past traffic generation patterns of IoT devices $\{x_i[m - s_p^i]\}_{i\in\mathcal{N}}^{p\in\{1,\ldots,|F_i|\}}$ for JFS under each forecasting scheme, and the input is the accumulated future traffic $\{x_i[m + k]\}_{i\in\mathcal{N}}^{k\in\{1,\ldots,K_i\}}$ (which is available only after the traffic generation patterns have been realized) for JFS under perfect forecasts.

The future traffic is not accessible to calculate the network performance under perfect forecasts since it has not yet been realized.

In addition, the results of the works [11, 12] have shown that network performance depends highly on the total number of IoT devices as well as the percentage of devices that fall in each class. Thus, taking into account this fact, the network performance emulator could be designed to have the number of devices of each class type as an input and the design should be irrespective of the network setup. Consequently, we develop an emulation by DAFS that will compute the network performance metrics based on $\mathbf{N}$, namely the vector of the instantaneous number of IoT devices in each device class in the coverage area of G.

5.1.2. *Emulation of the Relationship Between Device Class Composition and Network Performance*

The Network Performance Estimator of DAFS in Fig. 2 estimates the network performance vector. For simplicity, we shall take throughput as the network performance metric in order to describe the ANN. The elements of the vector $\hat{\eta}$ are $\hat{\eta}_{\text{ARIMA}}, \hat{\eta}_{\text{LSTM}}$ and $\hat{\eta}_{\text{MLP}}$. Recall that the elements of the vector $\mathbf{N}$ are $N_{\text{FBP}}, N_{\text{VBP}}, N_{\text{FBA}}$ and N_{VBA}. The Network Performance Estimator forms a relation between $\mathbf{N}$ and $\hat{\eta}$. In order to emulate this relation, we use an ANN as the Network Performance Estimator of DAFS, where the input of the ANN is $\mathbf{N}$, and the output is $\hat{\eta}$.

In this work, we use an MLP model for this ANN, which we train by using the backpropagation algorithm for the input-output pairs $(\mathbf{N}^v, \eta^v)$, where at each sample v, $\mathbf{N}^v$ denotes the realization of $\mathbf{N}$, and η^v denotes the corresponding η vector that

is collected via JFS.[6] Once the MLP has been trained, we use the resulting MLP for the Network Performance Estimator of DAFS in Fig. 2 in order to estimate $\hat{\eta}$ for any given network that contains at that instance the number of IoT devices in each IoT device class given by the vector $\mathbf{N}$.

5.1.3. Caveats in Using an ANN for Emulation

There are certain caveats in using an ANN, in particular an MLP, in learning the above relation. First, in training the MLP, it is entirely possible that the weights of the MLP converge to a local optimum in the training process. However, we note that if there is a sufficiently simple relationship between $\mathbf{N}$ and the network performance metrics, the probability that training converges to a local (rather than global) optimum is reduced. Second, we note that $\mathbf{N}$ gives only partial information on the network state: For example, the actual traffic generation patterns of all of the devices in C_G across all time would give a full description of the traffic offered.

The reason that we choose to input only $\mathbf{N}$ into the ANN, rather than actual traffic generation patterns of IoT devices, is that we would like the results to generalize to the case where new devices, whose traffic generation patterns the network has not seen, constitute the set of IoT devices in the network. In those cases, it is essential that the forecaster selection be made dynamically and automatically without any reliance on the actual traffic generation patterns of individual IoT devices. That is, by relinquishing full knowledge on the actual traffic generation patterns that lead to network performance, our DAFS gains the ability to generalize to new scenarios that involve IoT devices that the network has not seen. (This generalization ability shall be demonstrated in Section 6.2.)

5.2. Forecasting Scheme Selection

For each scheduling window of JFS, as soon as the Network Performance Estimator has formed its estimate, the Forecasting Scheme Selection block selects the best forecasting scheme in the forecaster of device i based on $\hat{\eta}$, which is estimated by the trained MLP in the Network Performance Estimator. As shown in Fig. 2, the Forecasting Scheme Selection is comprised of the *argmax* operator as well as the block that selects among ARIMA, LSTM, and MLP. The Forecasting Scheme Selection of DAFS returns the index of the scheme that achieves the highest $\hat{\eta}$, i.e. the forecasting scheme that is estimated to achieve the highest throughput for the current $\mathbf{N}$. After DAFS returns the best forecasting scheme, it switches the forecaster of device i to that scheme. That is, in Forecaster$_i$, the forecasting scheme, which is selected by DAFS, is used for computing $\{\hat{x}_i[m+k]\}_{k\in\{1,\ldots,K_i\}}$.

6. Results

6.1. Methodology

6.1.1. Data Collection Methodology

We now describe how we collected the dataset that we have used for our results on DAFS. First, we let the total number of devices N be a multiple of 10. Then, we consider all of the 4-partitions[7] of 10. We denote each such a 4-partition of 10 by $\tilde{\mathbf{N}}$, and we set $\mathbf{N} = (N/10)\tilde{\mathbf{N}}$. Then, for each $\mathcal{N}$, in order to compute the value of η, we invoke JFS under each of the ARIMA, LSTM and MLP forecasting schemes for 10 randomly selected scheduling windows and take the average over these windows. For the traffic generation pattern of individual IoT devices, we use the publicly available IoT traffic dataset [48].

6.1.2. Performance Evaluation Metric for DAFS

In order to measure the performance of DAFS, instead of using the existing error metrics in the machine learning literature (such as r^2, Mean Square Error (MSE), symmetric Mean Absolute Percentage Error (sMAPE)), we define an error metric, which measures how far the performance of JFS under DAFS is from the best achievable performance.

We shall let $\mathcal{X}$ denote a network performance metric (such as throughput η or energy consumption $\mathcal{E}$). Furthermore, let $\mathcal{X}_{\text{best}}$ denote the best network performance metric that is achieved by JFS over all forecasting schemes where $\mathcal{X}$ can be replaced

[6] The collection of the dataset is described in Section 6.1.1.

[7] A k-partition of a positive integer n is a vector of k positive integers that sum up to n.

with any network performance metric used in the JFS. Moreover, let $\mathcal{X}_{\text{worst}}$ denote the worst network performance metric that is achieved by JFS over all forecasting schemes. In addition, let $\mathcal{X}_{\text{DAFS}}$ denote the network performance metric that is achieved by JFS under DAFS.

We denote this error metric by $f_{\mathcal{X}}$ and define it as

$$f_{\mathcal{X}} \equiv \left| \frac{\mathcal{X}_{\text{DAFS}} - \mathcal{X}_{\text{worst}}}{\mathcal{X}_{\text{best}} - \mathcal{X}_{\text{worst}}} \right| \delta_{\mathcal{X}_{\text{best}} \neq \mathcal{X}_{\text{worst}}} \quad (1)$$

Above, δ_{Ψ} is defined to be 1 if Ψ is a true statement and is 0 otherwise. Accordingly, the range of $f_{\mathcal{X}}$ is $[0, 1]$, where 0 represents the minimum and 1 represents the maximum error that can be achieved by DAFS.

6.1.3. Cross-Validation

In order to evaluate the performance of DAFS, we use 10-fold cross-validation, which we implement by using the *scikit-learn* library [49] in Python.[8] For each test, we measure the error metric for throughput, namely f_{η}, which is calculated as in (1).

6.1.4. Parameter Tuning of the Network Performance Estimator Implemented as MLP

In this paper, we design the MLP in which the architectural parameters are the number of layers E, the number of neurons h_e at each layer e and the activation function at each neuron. For throughput, in order to achieve the best performance of DAFS where the Network Performance Estimator is implemented as an MLP, we select the best architectural parameters as follows:

Step 1: We generate 100 MLP architectures for each of which we set the architectural parameters as follows: First, we set $E = 5$, where the fifth layer is the output layer for which $h_5 = 3$, i.e. the total number of available forecasting schemes. The activation function of each neuron at $e = 5$ is set to the sigmoid function[9]. The activation function of each neuron at each hidden layer $e \in \{1, 2, 3, 4\}$ is set to the tangent hyperbolic function. Furthermore, we select the value of h_e for each layer $e \in \{1, 2, 3, 4\}$ randomly in the range $[2, 150]$ at increments of 2.

Step 2: For each run, for each fold of the cross-validation, we split the dataset in Section 6.1.1 into the training and the test set such that 90% of the samples are used in training and the remaining 10% in testing and the data samples for each set are randomly selected without replacement. Then, we train each of 100 MLP architectures in DAFS using the training set. By using each trained MLP, we measure the performance of DAFS with respect to f_{η} on the test set.

Step 3: Over 100 MLP architectures, we select the architecture that achieves the best performance, which is closest to 0 for the error metric f_{η} for throughput.

6.2. Performance Evaluation of DAFS

6.2.1. Performance of JFS under DAFS

We now present the performance of the JFS system under the DAFS method. It should be noted that the outputs of DAFS provide the throughput estimates for each of the forecasting schemes under examination; however, each output is computed under the assumption that a single forecasting scheme is used across all devices irrespective of the device class. (In this case, a single forecasting scheme is used not only for the execution but also for the training of DAFS.)

First, we examine the performance of JFS under DAFS with respect to the error metric f_{η}. In Fig. 4, we present the box plot of the cross-validation error with respect to the metric f_{η} for JFS under DAFS. For a fixed N, the performance variation in the box plot occurs due to the folds of cross-validation, where the red line corresponds to the median. In this figure, we see that although the median of the error metric f_{η} increases slightly with the number of devices, it remains below 0.2 for all values of N; that is, more than 80% of the gap between η_{worst} and η_{best} has been closed by DAFS. In addition, our results show that the maximum f_{η} over all folds of cross-validation across all N is approximately 0.23 (which occurs at $N = 800$). This indicates that JFS achieves at least 77% of its highest possible throughput under DAFS even when outliers are present.

[8] The 10-fold cross-validation method first splits the dataset into 10 parts where for each fold of this method, we train the DAFS on 90% of the dataset and test it on the remaining 10%.

[9] Since throughput takes values in [0, 1], the sigmoid activation function is suitable.

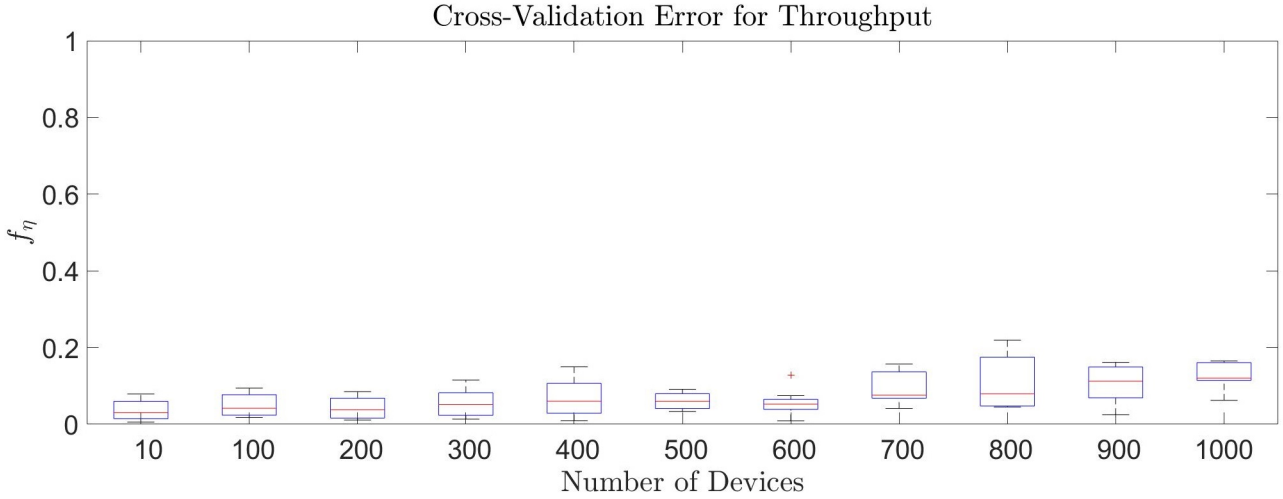


Figure 4: Box plot of the error in f_η for the throughput estimation performance of DAFS under 10-fold cross-validation as a function of the number of devices N in the coverage area.

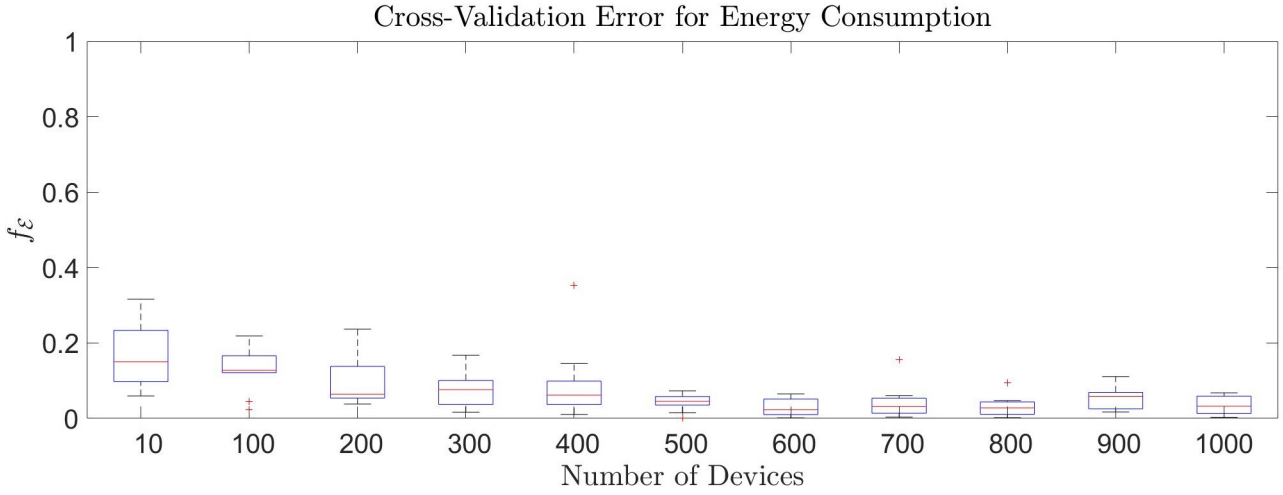


Figure 5: Box plot of the error in $f_\mathcal{E}$ for the energy consumption estimation performance of DAFS under 10-fold cross-validation as a function of the number of devices N in the coverage area.

Second, in Fig. 5, we present the cross-validation of the error of JFS under DAFS in energy consumption, denoted by $f_\mathcal{E}$, for the best MLP architecture. In this figure, we see that the median of $f_\mathcal{E}$ is under 0.2 and the maximum of $f_\mathcal{E}$ is under 0.40 for all values of N. We also see that the median of $f_\mathcal{E}$ is around 0.18 for $N = 10$, and decreases slowly up to $N = 500$. Between $N = 500$ and $N = 1000$, the median of the error is almost constant with fluctuations which are caused by the randomness in the dataset as well as cross-validation. In addition, we see a trend in the standard deviation that is similar to that of the median, which decreases up to $N = 500$ and remains almost constant for $N \geq 500$. Our results in Fig. 5 collectively show that the DAFS method successfully selects the best-performing forecaster with a relatively small error.

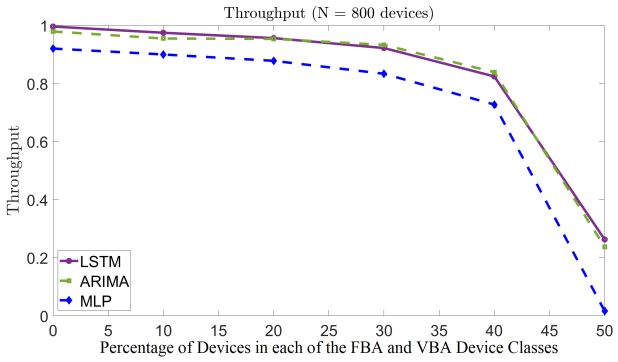
In conclusion, the results on throughput in Fig. 4 and energy consumption in Fig. 5 show that for all values of N, $10 \leq N \leq 1000$, the median of each of f_η and $f_\mathcal{E}$ is less than 0.2; hence, JFS almost achieves the best performance by selecting the forecaster via DAFS.

These results also indicate that $\hat{\eta}$ and $\hat{\mathcal{E}}$ (which are estimated by the Network Performance Estimator in DAFS) are close to η and $\mathcal{E}$, respectively, since the

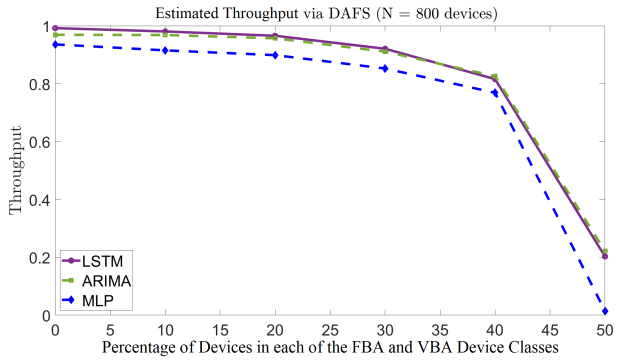
error in the estimaiton of each of η and $\mathcal{E}$ significantly affects the selection of the forecaster and thus the performance of JFS under DAFS.

6.2.2. *Performance of the Network Performance Estimator in DAFS*

We aim to examine the performance of MLP when it is used as the Network Performance Estimator of DAFS in order to estimate η. To this end, in this section, we present the results on an experiment in which the number of devices in the Aperiodic (FBA and VBA) classes increases. We set $N_{\text{FBA}} = N_{\text{VBA}}$ and $N_{\text{FBP}} = N_{\text{VBP}}$. Furthermore, we increase $(N_{\text{FBA}} + N_{\text{VBA}})/N$, while the value of N remains constant. Then, we present the comparison between the actual performance of the JFS under the LSTM, ARIMA and MLP forecasters and the estimate obtained by the Network Performance Estimator of DAFS.



(a) Actual throughput



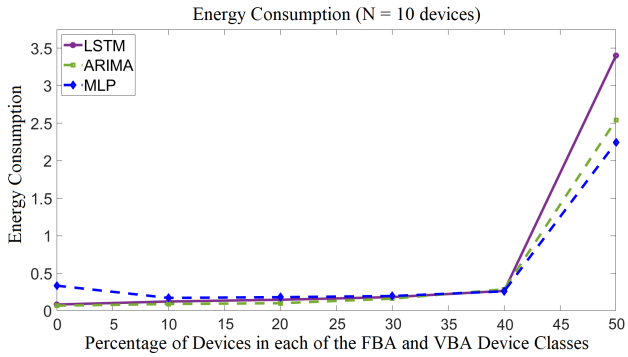
(b) Estimated throughput

Figure 6: (a) Throughput of JFS, η, under the LSTM, ARIMA and MLP forecasters, and (b) estimated throughput, $\hat{\eta}$, via the Network Performance Estimator of DAFS for $N = 800$ devices.

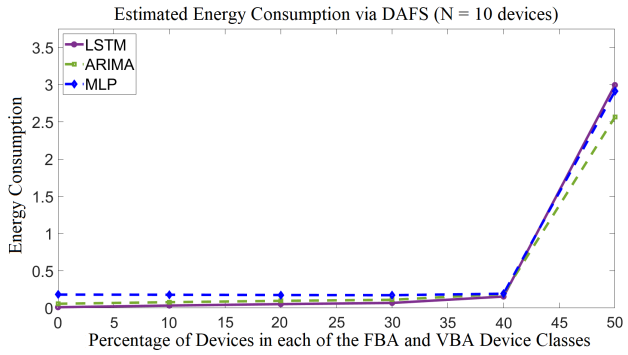
Fig. 6 shows the actual throughput η and the estimated throughput $\hat{\eta}$ for $N = 800$ for our experiment. Since the standard deviation and the maximum error of DAFS are shown to be the highest for 800 devices in Fig. 4, we shall analyze the throughput estimation performance of the Network Performance Estimator of DAFS for $N = 800$.

When we examine our results in Fig. 6(a) and Fig. 6(b), we see that the estimation of the throughput in Fig. 6(b) is highly accurate based on the actual throughput in Fig. 6(a). For example, the results in Fig. 6(b) show that the Network Performance Estimator in DAFS successfully captures the sharp decrease of the actual throughput in Fig. 6(a) at the point where the percentage of the number of devices in each of the Aperiodic classes equals 40%. In addition, the Network Performance Estimator of DAFS estimates the accurate order of the forecasters for the majority of the percentages except 30% and 50%. At these two percentages, although DAFS incorrectly estimates the order of LSTM and ARIMA, it does not significantly affect the performance of JFS since the throughput under LSTM is almost equal to that under ARIMA.

Fig. 7 shows the $\mathcal{E}$ in Fig. 7(a) and $\hat{\mathcal{E}}$ in Fig. 7(b) for $N = 10$ for the experiment that we describe above. Since the median and the standard deviation of DAFS are the highest for 10 devices for the CV results in Fig. 5, we compare the energy consumption and the estimate obtained by the Network Performance Estimator of DAFS for $N = 10$. In these figures, our results show that the estimations produced by the Network Performance Estimator of DAFS are considerably successful in terms of both the order of the forecasters and the magnitude of energy consumption. Although DAFS successfully captures the sharp increase at 40%, it incorrectly estimates the order of MLP and the ARIMA for percentages greater than 40%, where MLP achieves the minimum $\mathcal{E}$. However, in Fig. 7(a), we see that selecting ARIMA instead of MLP does not affect the energy consumption of the system significantly because the value of $\mathcal{E}_{\text{ARIMA}}$ is close to that of $\mathcal{E}_{\text{MLP}}$. In addition, selecting either ARIMA or MLP performs much better than selecting LSTM.



(a) Actual energy consumption



(b) Estimated energy consumption

Figure 7: (a) Energy consumption of JFS, $\mathcal{E}$, under the LSTM, ARIMA and MLP forecasters, and (b) estimated energy consumption, $\hat{\mathcal{E}}$, via the Network Performance Estimator of DAFS for $N = 10$ devices.

6.2.3. *Generalization Ability of DAFS with Respect to the Architectural Parameters of the Network Performance Estimator*

We now present the generalization ability of the DAFS method for JFS against the selection of the architectural parameters of the Network Performance Estimator for which we used MLP. We analyze how DAFS performs under randomly selected architectural parameters of MLP with respect to f_η. In this analysis, for each of the randomly generated 100 MLP architectures in Section 6.1.4, we perform 10-fold cross-validation as described in Section 6.1.3 and take the mean of the performance over the folds of the cross-validation.

In Fig. 8, we present the histogram of the performance of DAFS with respect to f_η over the MLP architectures. In this figure, we see that the mean error over the cross-validation folds for the performance of the DAFS method for JFS is between 0 and 0.1 over randomly created 100 MLP architectures. Thus, the performance of DAFS on throughput is shown to be highly robust with respect to the selection of the architectural parameters of the Network Performance Estimator.

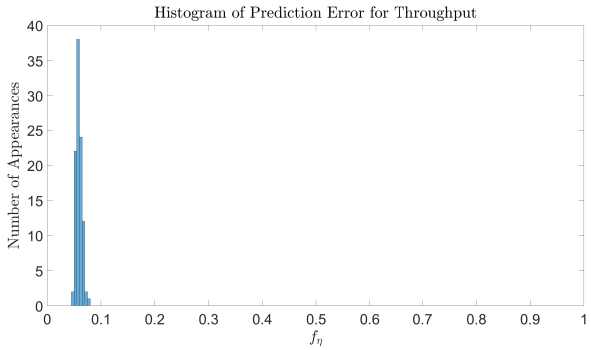


Figure 8: Histogram of the cross-validation throughput performance of DAFS over 100 randomly generated MLP architectures

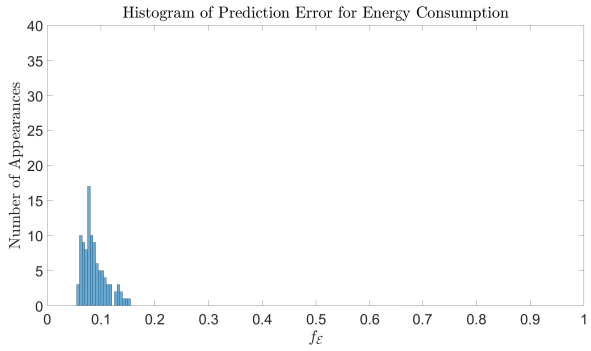


Figure 9: Histogram of the cross-validation performance of DAFS on the energy consumption over 100 randomly generated MLP architectures

In Fig. 9, we present the histogram of the performance error of JFS under the DAFS method for energy consumption as measured by $f_\mathcal{E}$ over MLP architectures. In this figure, we see that the mean of the $f_\mathcal{E}$ error over the cross-validation folds for the performance of the DAFS method for JFS is between 0.05 and 0.2 over randomly created 100 MLP architectures. The results in Fig. 9 show that the performance of DAFS on energy consumption is also highly robust with respect to the selection of the architectural parameters of the Network Performance Estimator.

6.2.4. *Performance of DAFS for Class-based Forecaster Selection*

We now evaluate the performance of the JFS system under DAFS in the specific case in which the forecasting scheme across the devices in each device class is identical yet the forecasting schemes across distinct device classes may be different. Recall that DAFS has been trained for the case in which JFS uses a single forecasting scheme across all devices irrespective of the class. By presenting this comparison, we aim to show how a single (trained) DAFS can be used for the selection of distinct forecasting schemes across the device classes.

We use the same DAFS trained for the case of a single forecasting scheme, as explained in Section 6.1.1. Then we execute DAFS in order to emulate the performance of JFS for a varying number of devices in each device class. We shall let "$\cdot$" denote the dot product of two vectors. In order to execute DAFS for each device class, we feed $\mathbf{N} \cdot \mathbf{v}$ (in the place of $\mathbf{N}$) into the Network Performance Estimator, where $\mathbf{v}$ denotes a four-element vector such that the value of each element of this vector equals 1 for the class under consideration and equals 0 otherwise. Then, these class-based forecasting schemes are applied to mixtures of devices across distinct class types.

For the performance evaluation of JFS under DAFS, we set $N_{\text{FBA}} = N_{\text{VBA}} = N_{\text{VBP}} = N/3$ and $N_{\text{FBP}} = 0$ for values of $N \in \{240, 480, 720, 960, 1200\}$. We also compare the throughput performance of DAFS under the above forecaster selection against that of a static selection of the best forecasting scheme based on an exhaustive search over all of the forecasting schemes under examination for each device class.

Fig. 10 presents a comparison of DAFS against the static method for class-based estimator selection. The results in this figure show that DAFS outperforms the static selection of forecasting schemes up to $N = 960$. As shown in the figure, the performance gap between DAFS and static selection decreases with N because DAFS is trained for the case in which JFS uses a single forecasting scheme for the entire bank of forecasters irrespective of the class. Accordingly, DAFS is practical in applications by virtue of the following facts: 1) DAFS is able to achieve a performance that is close to or better than that achieved by static selection of the forecasting schemes based on exhaustive search. 2) DAFS has a high performance and robustness, as shown in Section 6.2. 3) DAFS has low computational requirements.

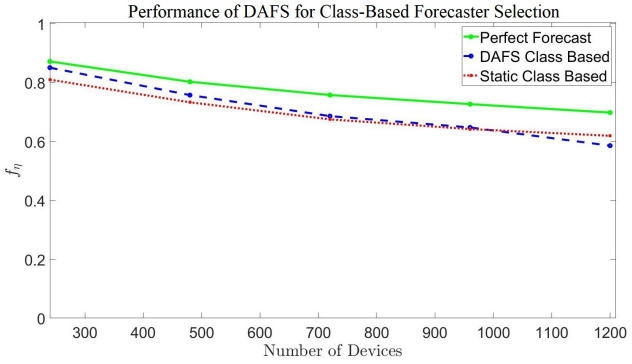


Figure 10: Comparison of the throughput of JFS under DAFS with that under perfect forecasts and static selection of the forecaster

6.3. *Training and Execution Time*

In this section, in order to demonstrate the potential of the DAFS method for practical applications, we present both the training and execution times of DAFS. We measured the execution time of the DAFS method on the Google Colab platform in the presence of a Tensor Processing Unit (TPU) accelerator.

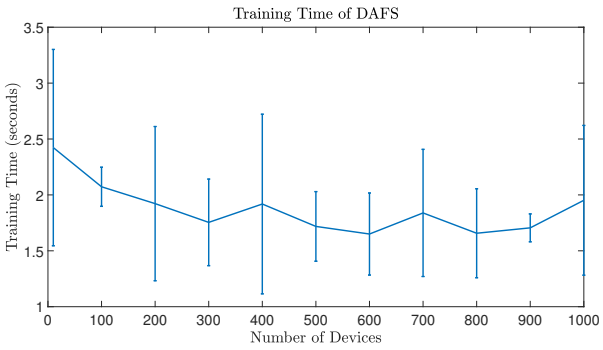


Figure 11: Mean of the training time of the DAFS method with standard deviation bars in terms of the number of devices in the coverage area.

Fig. 11 shows the mean of the training time of DAFS and the standard deviation of that over the training times, each of which is measured at each fold of the cross-validation (contains about 2200 training samples). In this figure, we see that the

mean training time of DAFS is around 2 seconds for almost all instances of the number of devices. The training time of DAFS does not significantly change with the number of devices since the number of training samples and the input-output sizes of the Network Performance Estimator are constant.

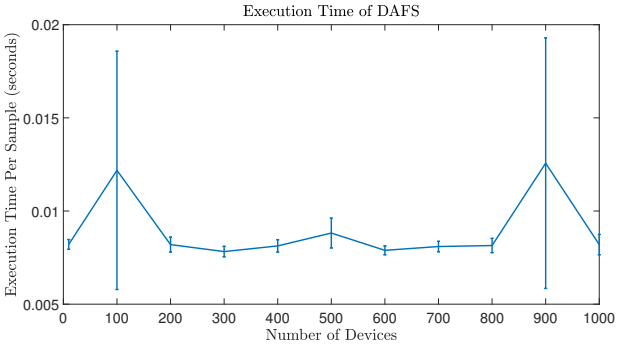


Figure 12: Mean of the execution time of the DAFS method per sample with standard deviation bars in terms of the number of devices in the coverage area.

Fig. 12 shows the mean of the execution time of DAFS and the standard deviation of that over the execution times, each of which is measured at each fold of the cross-validation. In this figure, we see that the mean of the computation time of DAFS fluctuates around 0.009 seconds and remains almost constant as the number of devices increases. The reason is that the length of each of $\mathbf{N}$ and η, which are the input and the output of the Network Performance Estimator in DAFS, respectively, does not depend on the number of devices and rather depends on the number of device classes and that of the available forecasting schemes. These results show that DAFS can easily be scaled up for any number of devices while incurring an acceptable penalty in computation time.

7. Conclusions

We have developed a novel method called Dynamic, Automatic Forecaster Selection (DAFS) for predictive solutions for the Massive Access Problem of IoT in the case where the number of IoT devices in the coverage area of an IoT gateway can vary dynamically.

The key novel idea in the design of DAFS is that the relationship between the current composition of the number of IoT devices in each device class and the resulting network performance can be emulated by an Artificial Neural Network. In this work, we showed that a Multi-Layer Perceptron architecture can successfully learn this relationship. Once this relationship is learned, the IoT gateway can choose on the fly which forecaster to use based on the observed numbers of IoT device classes that currently fall in the coverage area in order to maximize throughput. Our results have also shown that the DAFS method is robust with respect to the selection of the architectural parameters, which implies that DAFS can be used in real-time applications.

We have extensively evaluated the performance of DAFS for two main cases: 1) Only one forecasting scheme is selected for the entire bank of forecasters, and 2) separate forecasting schemes are selected for the device classes.

Our results have shown that the DAFS method is able to achieve at least 80% of the best performance on the average of 10-fold cross-validation results for both throughput and energy consumption.

These results indicate that the DAFS method significantly improves the performance of JFS over the methods that utilize a fixed forecasting scheme [11, 12, 13, 14].

We also showed that both robustness and fast calculation of DAFS make JFS practical to use in real-time applications, in which both static and mobile IoT devices are dynamically connected and disconnected from the IoT gateway. In addition, adapting to dynamic network conditions via the DAFS method is a significant step towards network performance provisioning for IoT.

References

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

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

[3] H. Jin, W. T. Toor, B. C. Jung, J.-B. Seo, Recursive pseudo-Bayesian access class barring for M2M communi-

cations in LTE systems, IEEE Transactions on Vehicular Technology 66 (9) (2017) 8595–8599.

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

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

[6] L. Tello-Oquendo, D. Pacheco-Paramo, V. Pla, J. Martinez-Bauset, Reinforcement learning-based ACB in LTE-A networks for handling massive M2M and H2H communications, in: 2018 IEEE International Conference on Communications (ICC), IEEE, 2018, pp. 1–7.

[7] N. Jiang, Y. Deng, A. Nallanathan, J. Yuan, A decoupled learning strategy for massive access optimization in cellular IoT networks, IEEE Journal on Selected Areas in Communications 39 (3) (2020) 668–685.

[8] A. O. Almagrabi, R. Ali, D. Alghazzawi, A. AlBarakati, T. Khurshaid, A Poisson process-based random access channel for 5G and beyond networks, Mathematics 9 (5) (2021) 508.

[9] M. Nakip, B. C. Gül, V. Rodoplu, C. Güzeliş, 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, 2019, pp. 102–109.

[10] M. Nakip, K. Karakayali, C. Güzeliş, V. Rodoplu, An end-to-end trainable feature selection-forecasting architecture targeted at the Internet of Things, IEEE Access 9 (2021) 104011–104028. doi:10.1109/ACCESS.2021.3092228.

[11] M. Nakip, V. Rodoplu, C. Güzeliş, D. T. Eliiyi, Joint forecasting-scheduling for the internet of things, in: 2019 IEEE Global Conference on Internet of Things (GCIoT), IEEE, 2019, pp. 1–7.

[12] V. Rodoplu, M. Nakıp, D. T. Eliiyi, C. Güzelis, A multi-scale algorithm for joint forecasting-scheduling to solve the massive access problem of IoT, IEEE Internet of Things Journal 7 (9) (2020) 8572–8589. doi:10.1109/JIOT.2020.2992391.

[13] V. Rodoplu, M. Nakip, R. Qorbanian, D. T. Eliiyi, Multi-channel joint forecasting-scheduling for the internet of things, IEEE Access 8 (2020) 217324–217354.

[14] M. Nakip, A. Helva, C. Güzeliş, V. Rodoplu, Subspace-based emulation of the relationship between forecasting error and network performance in joint forecasting-scheduling for the Internet of Things, in: IEEE World Forum on Internet of Things (WF-IoT), 2021, pp. 247–252.

[15] M. Nakip, E. Gelenbe, Randomization of data generation times improves performance of predictive IoT networks, in: IEEE World Forum on Internet of Things (WF-IoT), 2021, pp. 350–355.

[16] N. Flick, D. Garlisi, V. R. Syrotiuk, I. Tinnirello, Testbed implementation of the meta-MAC protocol, in: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2016, pp. 580–585.

[17] J. Ansari, X. Zhang, A. Achtzehn, M. Petrova, P. Mähönen, A flexible MAC development framework for cognitive radio systems, in: 2011 IEEE Wireless Communications and Networking Conference, IEEE, 2011, pp. 156–161.

[18] W. Hu, X. Li, H. Yousefi'zadeh, LA-MAC: A load adaptive MAC protocol for manets, in: GLOBECOM 2009-2009 IEEE Global Telecommunications Conference, IEEE, 2009, pp. 1–6.

[19] K.-C. Huang, X. Jing, D. Raychaudhuri, MAC protocol adaptation in cognitive radio networks: An experimental study, in: 2009 Proceedings of 18th International Conference on Computer Communications and Networks, IEEE, 2009, pp. 1–6.

[20] S. Hu, Y.-D. Yao, Z. Yang, MAC protocol identification approach for implement smart cognitive radio, in: 2012 IEEE International Conference on Communications (ICC), IEEE, 2012, pp. 5608–5612.

[21] W. Wang, C. Dong, H. Wang, A. Jiang, Design and implementation of adaptive MAC framework for uav ad hoc networks, in: 2016 12th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN), IEEE, 2016, pp. 195–201.

[22] M. Sha, R. Dor, G. Hackmann, C. Lu, T.-S. Kim, T. Park, Self-adapting MAC layer for wireless sensor networks, in: 2013 IEEE 34th Real-Time Systems Symposium, IEEE, 2013, pp. 192–201.

[23] A. Farago, A. D. Myers, V. R. Syrotiuk, G. V. Zaruba, A new approach to MAC protocol optimization, in: Globecom'00-IEEE. Global Telecommunications Conference. Conference Record (Cat. No. 00CH37137), Vol. 3, IEEE, 2000, pp. 1742–1746.

[24] C. Doerr, M. Neufeld, J. Fifield, T. Weingart, D. C. Sicker, D. Grunwald, Multimac-an adaptive MAC framework for dynamic radio networking, in: First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005., IEEE, 2005, pp. 548–555.

[25] J. Zhen, V. Rodoplu, Automated MAC protocol generation under dynamic traffic conditions, in: 2013 IEEE Global Communications Conference (GLOBECOM), IEEE, 2013, pp. 152–157.

[26] M. Qiao, H. Zhao, S. Wang, J. Wei, MAC protocol selection based on machine learning in cognitive radio networks, in: 2016 19th International Symposium on Wireless Personal Multimedia Communications (WPMC), 2016, pp. 453–458.

[27] S. Hu, Y. Yao, Z. Yang, MAC protocol identification using support vector machines for cognitive radio networks, IEEE Wireless Communications 21 (1) (2014) 52–60.

[28] S. Tomforde, J. Hähner, Organic network control: turning standard protocols into evolving systems, in: Biologically Inspired Networking and Sensing: Algorithms and Architectures, IGI Global, 2012, pp. 11–35.

[29] I. Tinnirello, D. Garlisi, F. Giuliano, V. R. Syrotiuk, G. Bianchi, MAC learning: Enabling automatic combination of elementary protocol components, in: Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization, 2016, pp. 89–90.

[30] M. Qiao, H. Zhao, S. Huang, L. Zhou, S. Wang, An intelligent MAC protocol selection method based on machine learning in wireless sensor networks., KSII Transactions on Internet & Information Systems 12 (11) (2018).

[31] S. Hu, Y.-D. Yao, Z. Yang, MAC protocol identification using support vector machines for cognitive radio networks, IEEE Wireless Communications 21 (1) (2014) 52–60.

[32] X. Zhang, W. Shen, J. Xu, Z. Liu, G. Ding, A MAC protocol identification approach based on convolutional neural network, in: 2020 International Conference on Wireless Communications and Signal Processing (WCSP), IEEE, 2020, pp. 534–539.

[33] H. Li, S. Peng, Z. Chen, X. Qin, MAC protocol recognition based on LSTM network in cognitive radio, Journal of Signal Processing 35 (5) (2019) 837–842.

[34] K. Andersson, Interworking techniques and architectures for heterogeneous wireless networks, Journal of Internet Services and Information Security (JISIS) 2 (1/2) (2012) 22–48.

[35] Y. Wu, X. Gao, S. Zhou, W. Yang, Y. Polyanskiy, G. Caire, Massive access for future wireless communication systems, IEEE Wireless Communications 27 (4) (2020) 148–156.

[36] X. Chen, D. W. K. Ng, W. Yu, E. G. Larsson, N. Al-Dhahir, R. Schober, Massive access for 5g and beyond, IEEE Journal on Selected Areas in Communications 39 (3) (2020) 615–637.

[37] N. Khambari, B. Ghita, L. Sun, Qoe-driven video enhancements in wireless networks through predictive packet drops, in: 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), IEEE, 2017, pp. 355–361.

[38] M. Li, X. Guan, C. Hua, C. Chen, L. Lyu, Predictive pre-allocation for low-latency uplink access in industrial wireless networks, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 306–314.

[39] J. Wen, M. Sheng, J. Li, K. Huang, Assisting intelligent wireless networks with traffic prediction: Exploring and exploiting predictive causality in wireless traffic, IEEE Communications Magazine 58 (6) (2020) 26–31.

[40] E. Gelenbe, M. Nakıp, D. Marek, T. Czachorski, Diffusion analysis improves scalability of IoT networks to mitigate the massive access problem, in: 2021 29th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2021, pp. 1–8.

[41] E. Soltanmohammadi, K. Ghavami, M. Naraghi-Pour, A survey of traffic issues in machine-to-machine communications over lte, IEEE Internet of Things Journal 3 (6) (2016) 865–884.

[42] C. Hoymann, D. Astely, M. Stattin, G. Wikstrom, J.-F. Cheng, A. Hoglund, M. Frenne, R. Blasco, J. Huschke, F. Gunnarsson, Lte release 14 outlook, IEEE Communications Magazine 54 (6) (2016) 44–49.

[43] S. Ali, N. Rajatheva, W. Saad, Fast uplink grant for machine type communications: Challenges and opportunities, IEEE Communications Magazine 57 (3) (2019) 97–103.

[44] M. Shehab, A. K. Hagelskjær, A. E. Kalør, P. Popovski, H. Alves, Traffic prediction based fast uplink grant for massive IoT, in: 2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications, IEEE, 2020, pp. 1–6.

[45] E. Eldeeb, M. Shehab, H. Alves, A learning-based fast uplink grant for massive IoT via support vector machines and long short-term memory, IEEE Internet of Things Journal (2021).

[46] A. R. Abdellah, O. A. K. Mahmood, A. Paramonov, A. Koucheryavy, IoT traffic prediction using multi-step ahead prediction with neural network, in: 2019 11th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), IEEE, 2019, pp. 1–4.

[47] C. Johnson, B. Khadka, E. Ruiz, J. Halladay, T. Doleck, R. B. Basnet, Application of deep learning on the characterization of tor traffic using time based features., J. Internet Serv. Inf. Secur. 11 (1) (2021) 44–63.

[48] IoT Traffic Generation Pattern Dataset (Jan 2021).
URL `https://www.kaggle.com/tubitak1001118e277/iot-traffic-generation-patterns`

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, the Journal of machine Learning research 12 (2011) 2825–2830.