Martín, J., Kondepu, K., de Vleeschaumer, D., Reddy, V., Guimaraes, C., Sgambelluri, A., Valcarengui, L., Papagianni, C. & Bernardos, C. J. (2022). Dimensioning V2N services in 5G networks through forecast-based scaling. *IEEE Access, 00*, 0000-0000.

DOI: 10.1109/ACCESS.2022.3142346

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Dimensioning V2N Services in 5G Networks through Forecast-based Scaling

JORGE MARTÍN-PÉREZ[1], KOTESWARARAO KONDEPU[2,5], DANNY DE VLEESCHAUWER[3], VENKATARAMI REDDY[4], CARLOS GUIMARÃES[1], ANDREA SGAMBELLURI[5], LUCA VALCARENGHI[5], CHRYSA PAPAGIANNI[6], CARLOS J. BERNARDOS[1]

[1]Universidad Carlos III de Madrid, Spain.
[2]Indian Institute of Technology, Dharwad, India.
[3]Nokia Bell Labs, Antwerp, Belgium.
[4]Indian Institute of Technology, Hyderabad, India.
[5]Scuola Superiore Sant'Anna, Pisa, Italy.
[6]University of Amsterdam, Netherlands.

Corresponding author: Jorge Martín-Pérez (e-mail: jmartinp@it.uc3m.es).

Work partially funded by the EU H2020 5GROWTH Project (grant no. 856709) and H2020 collaborative Europe/Taiwan research project 5G-DIVE (grant no. 859881).

ABSTRACT With the increasing adoption of intelligent transportation systems and the upcoming era of autonomous vehicles, vehicular services (such as remote driving, cooperative awareness, and hazard warning) will have to operate in an ever-changing and dynamic environment. Anticipating the dynamics of traffic flows on the roads is critical for these services and, therefore, it is of paramount importance to forecast how they will evolve over time. By predicting future events (such as traffic jams) and demands, vehicular services can take proactive actions to minimize Service Level Agreement (SLA) violations and reduce the risk of accidents. In this paper, we compare several techniques, including both traditional time-series and recent Machine Learning (ML)-based approaches, to forecast the traffic flow at different road segments in the city of Torino (Italy). Using the the most accurate forecasting technique, we propose n-max algorithm as a forecast-based scaling algorithm for vertical scaling of edge resources, comparing its benefits against state-of-the-art solutions for three distinct Vehicle-to-Network (V2N) services. Results show that the proposed scaling algorithm outperforms the state-of-the-art, reducing Service Level Objective (SLO) violations for remote driving and hazard warning services.

INDEX TERMS Vehicle-to-Network, Scaling, Forecasting, Time-series, Machine Learning

I. INTRODUCTION

The 5th generation (5G) of mobile communications revisits the traditional design of cellular systems that focused on connectivity, towards the support of a wide variety of network services supporting a disparate set of requirements and capabilities in a shared physical infrastructure. To offer such distinct services, network operators' infrastructure is significantly changing, with 5G networks shifting from the monolithic architecture of previous generations to a highly modular, highly flexible, and highly programmable architecture. Network Function Virtualization (NFV) and Software-Defined Networking (SDN), along with the convergence of mobile networks, Edge and Cloud technologies, are key enablers for realizing such vision. In doing so, a custom-fit paradigm emerges where virtual and isolated networks (the so-called network slices [1]) are provided over the same and shared infrastructure and tailored to particular network services and their requirements.

Managing the lifecycle of such network services is a critical aspect of efficient service delivery in 5G. First, network services and their corresponding network slices (hereinafter referred only as services) must be orchestrated on-demand. This step requires the initial dimensioning of the service and relies mostly on pre-defined information. Second, service elasticity is required, adapting the system to workload changes in order to avoid any degradation of the service performance and violation of Service Level Agreements (SLAs). To this end, traditional scaling approaches include static

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

or reactive (e.g., threshold-based) solutions. However, they are incapable of facing unforeseen events, especially when multiple services must coexist over the same infrastructure. Consequently, service providers typically over-provision critical resources (i.e., network, computing) which increases the cost of service provisioning and reduces the number of services that can be supported simultaneously over the same shared infrastructure.

An efficient allocation of resources to coexisting services is essential in order to maximize the utilization of such resources while reducing service costs. Traffic forecasting may constitute a key component, aiding orchestrators and management entities in their decision-making processes by estimating the future demand of running services. Thus preemptive scaling actions (e.g., scaling in/out or up/down) can be taken to accommodate the expected demand beforehand, taking also into consideration the actual time required to scale the service.

Vehicular-to-Network (V2N) services constitute a set of upcoming use cases that can benefit from forecasting incoming vehicular traffic to continuously meet their strict reliability and low-latency constraints. At the same time, it allows network, storage and computing resources to be scale accordingly in a pro-active fashion. However, forecasting real-time traffic information is not a straightforward task due to unexpected events like e.g., car accidents. Nevertheless, there are existing techniques (see Section II-B) that propose traditional time-series and Machine Learning (ML)-based methods to forecast vehicular traffic based on its periodic patterns.

In this paper, we address network service elasticity through vehicular traffic forecasting, and contribute to the state-of-the-art as follows:

- We formulate the V2N service scaling as an optimization problem using queuing theory to derive V2N service delays.
- We compare several forecasting techniques, testing their performance on a vehicular traffic dataset for the city of Torino (Italy), before and after the COVID-19 lockdowns.
- We propose an online training approach to update the prediction on-the-fly, showing how it improves the accuracy of forecasting techniques.
- We propose a scaling algorithm, denoted as n-max scaling, to solve the V2N scaling problem using a forecasting techniques and assisted by the proposed online training.
- We perform a comparison of the proposed n-max scaling algorithm against existing state-of-the-art solutions, demonstrating its feasibility with respect to V2N service scaling for remote driving, cooperative awareness, and hazard warning services.

The remainder of this paper is organized as follows: Section II discusses the related work on forecasting techniques and their application for forecasting road traffic, and network or service dimensioning purposes. Section III presents the considered system model based on queuing theory and formulates the scaling problem to be solved. Section IV describes several techniques to forecast road traffic and evaluates their performance using a road traffic dataset. Subsequently, Section V describes how existing state-of-the-art algorithms solve the formulated V2N scaling problem, and presents the n-max algorithm and its performance on different V2N services with strict latency constraints. Finally, Section VI discusses the main findings of this study and points out future research directions.

II. BACKGROUND AND RELATED WORK

This Section refers to state-of-the-art forecasting techniques for *(i)* road traffic and *(ii)* network traffic flows. Moreover it provides an overview of existing works on forecasting methods used to support network service elasticity.

A. FORECASTING TECHNIQUES

As stated in [2], the *"every-day life presents countless situations where one must somehow estimate what will happen in the future, as a basis for reaching a decision or taking action"*. Such estimation can also be interpreted as a prediction or forecast.

Traditionally, forecasting techniques involve time series methods, such as Error, Trend, Seasonality (ETS), Auto-Regressive Integrated Moving Average (ARIMA) [3], and Triple Exponential Smoothing (TES) (i.e., Holt-Winters) [2]. These methods usually require a limited number of computational resources and low energy because they are mainly based on simple analytical formulas [4].

With the fast growth of available datasets, forecasting approaches started to adopt ML techniques, such as Long Short-Term Memory (LSTM) [5] and Recurrent Neural Networks (RNN). In other words, ML is empowering forecasting techniques with the means to implement complex multivariate analysis, accounting for different factors that impact a specific phenomenon. However, in contrast to traditional time series techniques, ML-based forecasting require a large number of resources and energy, especially for training, which might limit their effectiveness. A careful evaluation of the tradeoff between cost and benefits of utilizing traditional time series versus ML-based techniques must be conducted [6] [7], before applying them to any specific scenario.

B. ROAD TRAFFIC FORECASTING

Forecasting techniques have been widely used in road traffic scenarios since they follow a periodic and variable pattern over time. However, time-series associated with road traffic also present some irregularities that make forecasting a challenging task. In particular, events as vehicle accidents may break the periodicity of the traffic time-series, and will detriment the forecasting accuracy; for it is difficult to predict the the accident itself, the number of involved vehicles, or even how many vehicles will be in congestion due to the traffic.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

Other situations may lead to time-series irregularities difficult to predict, for example, concerts, road maintenance jobs, or bank holidays that cause traffic jams in the city limits, etc. Forecasting situations as the aforementioned is a challenging task, especially because the associated time- series irregularities – like a bump in the traffic flow due to an accident – are rarely foreseen in the data used to train the algorithm. Traffic forecasting algorithms should consider these artifacts in the road traffic time-series, and cannot assume a periodic and stable pattern over time; but rather, a time-series with irregularities due to the cited unexpected events. Hence, it is a challenge to design an algorithm to accurately forecast both the traffic pattern and its unexpected irregularities.

Traditional time-series were firstly adopted to forecast road traffic flows, with methods such as ETS, ARIMA, and TES (i.e., Holt-Winters) [6] [8]. With the emergence of ML, works such as [9] and [10] respectively applied, for the first time, Stacked AutoEncoders (SAEs) and Restricted Boltzmann Machine (RBM) models to forecast road traffic flows. In [11], a deep regression model with four layers (including one input, two hidden, and one output layers) is used to forecast vehicle flows in a city. Other works rely on the utilization of LSTM [12] [13], Deep Belief Network (DBN) [14], Dynamic Fuzzy Neural Networks (D-FNN) [15], and Gated Recurrent Units (GRU) [16], showing promising results on the application of ML-based techniques for road traffic forecasting.

C. FORECASTING APPROACHES FOR ELASTIC NETWORK SERVICES

Forecasting techniques are also used in telecommunication networks to ease and automate tasks related to the lifecycle management of networks and services. As an example, predictive analytics is a key component of the Zero touch network & Service Management (ZSM) framework envisioned by ETSI [17], as an alternative to static rule-based approaches, which are inflexible and hard to manage.

In [18], deep artificial neural networks are used to forecast network traffic demands of network slices with different behaviors. Similarly, in [19], a Holt-Winters-based forecasting analyzes and forecasts traffic requests associated with a particular network slice, which is dynamically corrected based on measured deviations. While the former proactively adapts the resources allocated to different services, the latter implements an admission control algorithm to maximize the acceptance ratio of network slice requests. In [20], LSTM is used by a dynamic bandwidth resource allocation algorithm, aiming to compute the best resource allocation to reduce packet drop probability.

A dynamic dimensioning of the Access and Mobility management Function (AMF) in 5G, which relies on traffic load forecasting using Deep Neural Network and LSTM, is proposed in [21] and [22]. In doing so, scaling decisions can be anticipated, avoiding the increase of the attachment time of user equipment and the percentage of rejected requests. A similar solution is also proposed in [23] targeting a dynamic and proactive resource allocation to the AMF, where LSTM, Convolutional Neural Networks (CNN), and a combined CNN-LSTM are used to forecast the traffic evolution of a mobile network.

There are also some works related to allocation of network resources for V2N services, using forecasting techniques. In π-ROAD [24], a deep learning architecture is proposed based on LSTM layers and autoencoders [25] to detect traffic events along a highway covered with an LTE deployment. The authors use the architecture also to predict future events, and formulate an optimization problem that allocates transmission blocks to an emergency slice associated to vehicular services as autonomous driving. Other works, such as [26], use an LSTM Neural Network to forecast the incoming vehicular traffic derived from a simulation, to perform the scaling and the migration of vehicular service instances in MEC platforms. The authors propose an algorithm called AutoMEC, that decides the migration and scaling based on the accuracy of the prediction and the load of neighboring stations.

The use of forecasting to tackle V2N service scaling is recent, given the late arise of applications as remote driving. Indeed, the literature typically assesses the scaling of V2N services [27], [28] with threshold-based mechanisms. However, even the papers that use forecasting for V2N scaling do not include a comparison of time series analysis and ML-based techniques. Moreover their performance is not assessed for scaling operations of V2N services with real road traffic traces. Such a scenario can highly benefit from the traffic forecasting techniques in Section II-B to *(i)* adapt to changing road traffic conditions (e.g., the COVID-19 lockdown witnessed in 2020); and *(ii)* scale vehicular services efficiently. This work addresses both challenges and, ultimately, paves the way for a scaling solution applied to vehicular services with strict end-to-end (E2E) delay requirements.

III. SYSTEM MODEL

We consider a 5G network infrastructure, with vehicles sending V2N application traffic to a Next Generation NodeB (gNB) located along the road. The gNB forwards packets to an edge server connected to an access ring switch (see [29] and [30] for the reference infrastructure). Packets are queued at the edge server and then processed by any of the CPUs allocated to the V2N application. In the example illustrated in Figure 1, two (blue) CPUs are allocated for V2N traffic processing. However, if the traffic demands a new (red) V2N application, users cannot be satisfied by the current configuration, thus the edge server scales vertically.

For the sake of tractability, we assume that vehicles arrive at the road segment covered by the gNB following a Poisson process with arrival rate λ_t. The arrival rate is time dependant t because it is expected that the number of vehicles on the road vary during the day. For example, there will be more vehicles during rush hours than very early in the morning.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

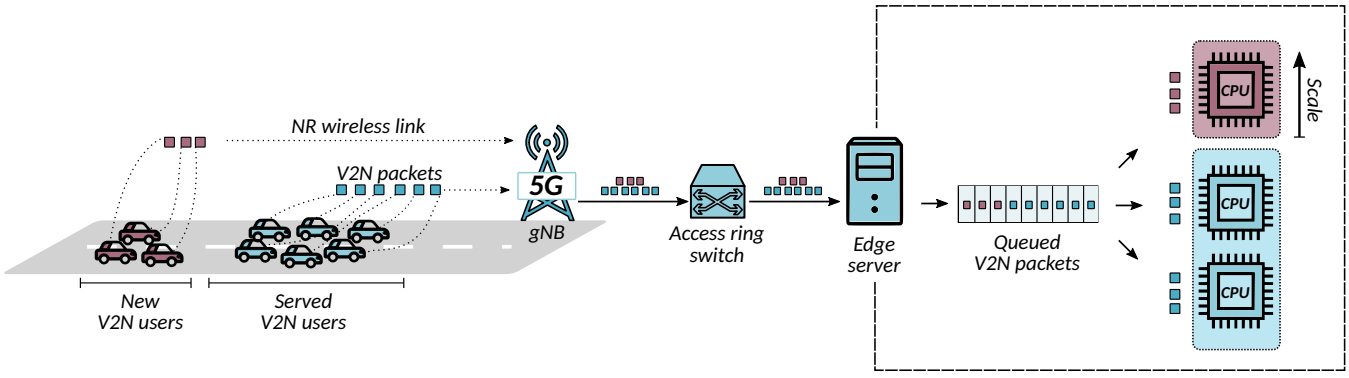


FIGURE 1: System Architecture[1]

The New Radio (NR) wireless link is assumed to use a numerology with 15 kHz Sub-Carrier Spacing (SCS) a frequency range in between 410 MHz and 7.125 GHz, normal cyclic prefix, 14 symbols per slot, maximum carrier bandwidth of 50 MHz, and a slot duration of 1 ms [31]. Based on [32] and the chosen numerology, packets are sent in a 1 ms transmission slot, rather than using the whole 10 slots transmission frame.

The edge server processes the incoming V2N application packets using any of the c CPUs allocated. The processing time of each CPU follows a Poisson distribution with rate μ. Thus, the scenario in Figure 1 is modeled using a $M/M/c$ queue [33]. Depending on the number of CPUs c_t and the arrival rate of vehicles λ_t at time t, the V2N application may or may not satisfy service requirements (in this case latency constraints).

Since the vehicles arrive according to a Poisson distribution and CPUs' processing time is also Poissonian, the average sojourn time of a V2N packet at time t is expressed as:

$$T_t = \frac{1}{\mu} + \frac{P_{Q,t}}{c_t\mu - \lambda_t} \tag{1}$$

where $P_{Q,t}$ is the probability that a V2N packet, arriving at time t, has to wait in the queue because the c_t allocated CPUs are busy. The expression of $P_{Q,t}$ is provided by the Erlang C formula:

$$P_{Q,t} = \frac{p_0(c_t\rho_t)^{c_t}}{c_t!(1-\rho_t)} \tag{2}$$

where $\rho_t = \frac{\lambda_t}{c_t\mu}$. The probability of having zero packets in the queue at the edge server at time t is

$$p_{0,t} = \left[\left(\sum_{n=0}^{c_t-1}\frac{(c_t\rho_t)^n}{n!}\right) + \frac{(c_t\rho_t)^{c_t}}{c_t!(1-\rho_t)}\right]^{-1} \tag{3}$$

The average sojourn time (Eq. 1), provides us with the number of CPUs c_t required to satisfy latency constraints of V2N services. This paper solves the following optimization problem of deciding how many CPUs c_{t+n} (and so the corresponding future λ_{t+n} demand) are required to satisfy the V2N latency constraints.

Problem III.1. *Given a latency constraint T_0, and a look-ahead value $n \in \mathbb{N}^+$, find a function $f : \mathbb{R}^N \mapsto \mathbb{N}^+$ that solves the optimization problem:*

$$\min_{\{c_t\}} \quad \sum_t c_t, \tag{4}$$

$$s.t. \quad c_{t+n} = f(\lambda_t, \lambda_{t-1}, \ldots, c_t, c_{t-1}, \ldots) \tag{5}$$

$$T_{t+n} \leq T_0 \tag{6}$$

In Section V we propose a vertical scaling algorithm, denoted as *n-max* to tackle the Problem III.1. The proposed algorithm forecasts the future traffic demand λ_{t+n} and scales up to c_{t+n} CPUs to meet the delay requirements. Before going into details on the *n-max* algorithm, we compare existing forecasting techniques in order to assess the best technique to be used in the proposed *n-max* scaling algorithm.

IV. COMPARISON OF FORECASTING TECHNIQUES

This Section provides a brief description of selected forecasting techniques and how *offline* and *online training* can be implemented, followed by an analysis of their performance using real road traffic traces.

A. SELECTED FORECASTING TECHNIQUES

In the scope of this work, distinct time series analysis and ML-based techniques are selected, namely Double Exponential Smoothing (DES) and Triple Exponential Smoothing time series techniques, Hierarchical Temporal Memory (HTM), Long Short-Term Memor (LSTM), Gated Recurrent Unit (GRU), Temporal Convolutional Networks (TCN), and Convolutional LSTM (TCNLSTM) ML-based techniques.

Although any time-series forecasting technique applies to assess the road traffic prediction, we resort to DES and TES based on their high performance in Edge and Cloud predictive analytics [7]. Moreover, DES and TES are great

[1]Currently served V2N packets are queued, and latter processed at any of the (blue) CPUs of an edge server. V2N packets are sent over a 5G gNB, and traverse an access ring switch before reaching the edge server. To accommodate the demand of new V2N users (red), the edge server scales and uses an additional (red) CPU.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

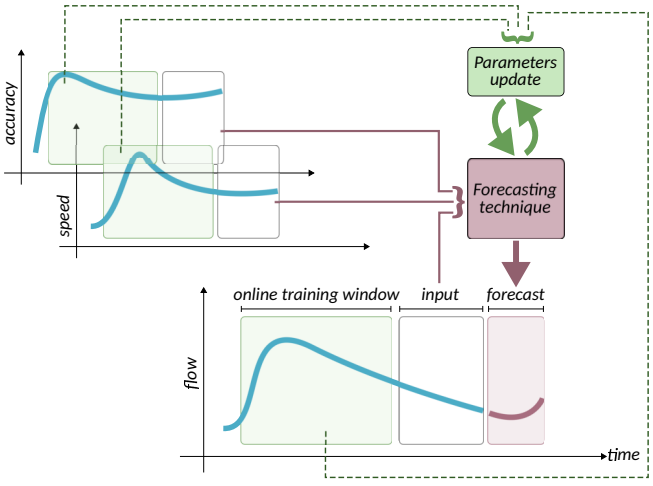


FIGURE 2: *Online training* for traffic forecasting

candidates given the periodicity/seasonality observed in road traffic time-series. Based on prior work in the state-of-the-art [26], we also consider LSTMs to forecast the road traffic and latter trigger V2N scaling. And with the goal of achieving higher accuracies, we also investigate a variation of LSTM using time convolution TCNLSTM, for the time convolution may allow extracting useful time patterns. Since we try TCNLSTM, we also give a chance to a plain TCN network without LSTM units, just to check if the time convolution on its own is enough to perform adequate forecasting. Last, we investigate memory-based ML solutions as HTM and GRUs that may succeed in saving representative events foreseen in the training stage, e.g., sudden increases of traffic.

The above forecasting techniques are analyzed considering two types of training: *(i)* an *offline training*, in which forecasting techniques learn their parameters in the training set; and *(ii)* an *online training*, where the parameters are also updated as the forecasting happens (see Figure 2). In this work, the *online training* uses a moving window (called online training window) comprising the most recent events, which are used to update its parameters before forecasting.

The next paragraphs provide an explanation of the selected forecasting techniques, their parameters, and how they are updated in the online/offline training stages:

1) ***Double Exponential Smoothing (DES) [2]:*** DES is a forecasting technique based on time series analysis. DES uses a smoothing time scale with *(i)* a *smooth* parameter; and *(ii)* a *trend* parameter. The smoothing value is obtained based on the previous value of *smooth* and *trend*. In DES, the *smooth* and *trend* parameters are learned during the *offline training* stage. If DES is evaluated using *online training*, the *smooth* and *trend* values are also updated using the *online training* window per forecast.
2) ***Triple Exponential Smoothing (TES) [2]:*** TES is another time series analysis technique. It exploits three different forecasting parameters, namely *(i) smooth*; *(ii) trend*; *(iii)* and *seasonality*. In TES, *offline training* is performed by calculating *smooth*, *trend*, and *seasonality* using the training set. Whereas in *online training*, the *smooth*, *trend*, and *seasonality* are updated for every forecast using the online training window.
3) ***Hierarchical Temporal Memory (HTM) [34]:*** The core component of the HTM forecaster is a temporal memory consisting of a two-dimensional array of cells that can either be switched on or off and that evolves with time. Cells can influence each other via *(i) synapses* and *(ii) update rules*. The *offline training* involves adjusting the *synapses* in such a way that the output bit strings resemble the actual input bit strings as much as possible. In that way, the temporal memory learns to forecast the next sparse bit strings based on the patterns in the sequence of input bit string it saw. The *online training* also updates the *synapses* using the *online training* window.
4) ***Long Short-Term Memory (LSTM) [5]:*** LSTM is a special form of Recurrent Neural Network (RNN) that can learn long-term dependencies based on the information remembered in previous steps of the learning process. It consists of a set of recurrent blocks (i.e., memory blocks), each of the block contains one or more memory cells, and multiplicative units with associated weights, namely, *(i) input*; *(ii) output*; and *(iii) forget gate*. LSTM is one of the most successful models for forecasting long-term time series, which can be characterized by different hyper-parameters, specifically the number of hidden layers, the number of neurons, and the batch size. For the *offline training* approach neurons' weights are updated running the back-propagation-through-time [35] over a training dataset. If LSTM uses *online training*, neurons' weights use the *online training* window to update their values using back-propagation-through-time.
5) ***Gated Recurrent Unit (GRU) [36]:*** Gated Recurrent Units (GRUs) are neurons used in RNNs and, as LSTMs cells, they store a hidden state that is recurrently fed into the neuron upon each invocation. Each neuron uses two gates, namely, *(i)* the *update gate*, and *(ii)* the *reset gate*. The former gate is an interpolator between the previous hidden state, and the candidate new hidden state; whilst the latter gate decides what to forget for the new candidate hidden state. GRUs keep track of as much information as possible of past events. Thus, their use in time-series forecasting is becoming popular in current state-of-the-art. Regarding the *offline/online training*, GRU works as the aforementioned LSTM.
6) ***Temporal Convolutional Networks (TCN) [11]:*** TCNs are deep learning architectures based on performing a temporal convolution over the input. The implemented version consists of two hidden layers, namely *(i)* a first layer to perform the temporal convolution; and *(ii)* a second layer to readjust the dimension of the convolution output. In particular, the convolution layer has a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

TABLE 1: Forecasting Features

Feature	Name	Values	Description
ϕ_1	flow	integer	vehicles/hour
ϕ_2	accuracy	$\{0,\ldots,100\}$	percent accuracy of the reported measurement
ϕ_3	speed	float	average vehicles' speed (km/hour)
ϕ_4	distance to Corso Orbassano	[0,35]	distance to Corso Orbassano road probe (km)
ϕ_5	day_of_week	$\{1,\ldots,7\}$	day of the week
ϕ_6	month	$\{1,\ldots,12\}$	month of the measurement
ϕ_7	day	$\{1,\ldots,31\}$	day of the measurement
ϕ_8	year	integer	year of the measurement
ϕ_9	hour_min	[0,24)	hour+minute/60

window size that is a fourth of the input length in the time domain. Both the *online* and *offline training* update the weights of the densely connected layers, and follow the same training procedure as LSTM.

7) ***Convolutional LSTM (TCNLSTM) [37]:*** In the Convolutional LSTM, both TCN and LSTM models are combined into a single unified framework. The input features are initially given to TCN layers. Then, the TCN layer output is fed to the LSTM layer. Lastly, the LSTM output feeds a final dense layer to produce the forecasting output. This model blends both the feature extraction of TCN layers and the memory of LSTM cells. In [38], it is shown that the LSTM performance can be improved by providing better features. Indeed, TCN helps by reducing the frequency variations in the input features. In this work, TCNLSTM is trained as LSTM for both in the *offline* and *online training*.

B. PERFORMANCE EVALUATION

In order to evaluate the performance of the techniques described above, a real road traffic dataset was collected from 28/01/2020 to 25/03/2020. The dataset comprises measurements from more than 100 road probes in the city of Torino (Italy), reporting their location, traffic flow, and vehicles speed. This dataset encompasses data pre- and post lockdown due to *COVID-19*.

Each forecasting technique is used to forecast the vehicles/hour traffic flow λ_t seen at Corso Orbassano road probe[2] at time t. The dataset includes a set of features ϕ_i reported by road probes s_j $(s_1,\ldots,s_{92})$. The numerical value of a feature reported by a probe at instant t is denoted as $x_t^{\phi_i,s_j}$. Table 1 enumerates the features ϕ_i, $i=\{1,\ldots,9\}$ used by the selected techniques. The dataset granularity is of 5 min., and throughout this paper $t+1$ represents the instant $t+5$ min.

[2]This is the road probe with the highest number of reported measurements in Torino (Italy).

Among all analyzed techniques, some of them can incorporate all features of past events to forecast the future flow of Corso Orbassano road. Thus, they take as input a matrix containing every feature reported by a road probe during the last h timestamps:

$$X_{t,h}=\begin{pmatrix} x_{t-1}^{\phi_1,s_1} & \ldots & x_{t-1}^{\phi_9,s_1} \\ \vdots & \ddots & \vdots \\ x_{t-1}^{\phi_1,s_{92}} & \ldots & x_{t-1}^{\phi_9,s_{92}} \\ \\ \vdots & \vdots & \vdots \\ \\ x_{t-h}^{\phi_1,s_1} & \ldots & x_{t-h}^{\phi_9,s_1} \\ \vdots & \ddots & \vdots \\ x_{t-h}^{\phi_1,s_{92}} & \ldots & x_{t-h}^{\phi_9,s_{92}} \end{pmatrix} \quad (7)$$

Since the dataset contains periods of *COVID-19* and *non-COVID-19*, it is divided into two parts, each with its training and testing sets, namely:

- *non-COVID-19 scenario:*
  - training: 28th January - 28th February
  - testing: 29th February - 07th March
- *COVID-19 scenario:*
  - training: 06th February - 07th March
  - testing: 8th March - 15th March

For the performance evaluation, *offline training* uses only the training sets to learn the weights/parameters, while *online training* also updates the learned weights/parameters using the testing sets and the online training window.

The selected techniques of Section IV-A were implemented using Python and the TensorFlow library. LSTM and TCN use the whole feature matrix $X_{t,h}$ to derive the predictions, while the other techniques just use the traffic flow feature. Table 2 summarizes the parameters that allowed to get the lowest Root Mean Square Error (RMSE) for each forecasting technique in the following experiments.

TABLE 2: Evaluation Parameters

Parameter	Forecasting techniques	Value
Level factor (α)	DES, TES	0.5, 0.5
Trend factor (β)	DES, TES	0.001, 0.001
Seasonality factor (γ)	TES	0.001 (3 days)
Hidden layers	TCN, LSTM, GRU, TCNLSTM	2,2, 1, 4
Neurons in hidden layer	TCN, LSTM, GRU, TCNLSTM	100
Epochs	TCN, LSTM, GRU, TCNLSTM	100
History window size (h)	TCN, LSTM, TCNLSTM	60 min.
	GRU	120 min.
Batch size	TCN, LSTM, GRU	5
Temporal memory	HTM	32x2048
Encoder representation		1024 bit str

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

In the following, we compare the performance of Section IV forecasting techniques as we increase the look-ahead time in the predictions, i.e., the number of future traffic flow values to predict. This analysis is of special importance given the time required to reconfigure and allocate the resources for a given virtualization technology, or type of service. That is, in case a service takes more than 5 min. to scale/instantiate, it is important to predict the demand 5 min. ahead to scale/instantiate on time. Results in Figure 3 illustrate how increasing the *look-ahead time* forecast leads to an increasing RMSE for every type of training (i.e., *online* and *offline* training) and dataset combinations (*COVID-19* and *non-COVID-19* scenario), as it becomes more difficult to forecast the traffic further in the future.

Figures 3a and 3b show the RMSE values of *offline training* in *non-COVID-19* and *COVID-19* scenarios. It can be observed that the HTM technique does not outperform a sample-and-hold benchmark, i.e., assume that the traffic in the next timestamp will be equal to the traffic in the current timestamp. Moreover, in the *online training* scenarios, it yields the worst performance. For the rest of the techniques, the ML-based approaches achieve the best performance for *offline training*. DES is not capable of capturing the trend, and the TES only does not capture the trend in the *COVID-19* scenario (see Figure 3b). Unlike DES and TES, ML-based techniques can capture the evolving traffic trend thanks to the update of their hidden states (apart from the TCN). This explains why ML-based techniques achieve lower RMSE when using offline forecasting. Furthermore, Figure 3a shows that DES technique has the highest RMSE values as the smooth and the trend values initially calculated during training are not updated in the testing phase. The other time-series technique (i.e., TES) mitigates the problem since its seasonality factor can capture the trend. Figure 3b shows the RMSE values of the considered techniques in *offline training* with *COVID-19* traffic. The considered scenario does not show any seasonality during 8$^{\text{th}}$ Mar - 15$^{\text{th}}$ Mar due to the *COVID-19* lockdown. Thus, the obtained TES results exhibit the highest RMSE value compared to all other techniques. This behavior is discussed later in this section.

Figure 3c and Figure 3d show the RMSE values of *online training* in *non-COVID-19* and *COVID-19* scenarios. The TES method outperforms all considered ML-based techniques even when the *look-ahead time* increases. In addition, results show that TES does not increase the RMSE as much as the ML-based techniques. This is due to the fact that it captures faster the new trends of traffic over time. Thus, the long *look-ahead time* forecasts are better as smoothing, trend, and seasonality are updated for every data point in the test set. Even though the traditional time series techniques (i.e., DES/TES) are limited to univariate time series, the *online* update of their parameters achieve a better performance than the ML-based techniques that account for all features reported in Table 1.

Finally, Figure 4 shows the real and the forecasted road traffic flow as a function of time. Here, the *look-ahead time* is set to 5 min., and *offline training* is used to forecast the traffic flow during the *COVID-19* scenario (i.e., same conditions as in Figure 3b). Figure 4 shows how the real traffic flow exhibits a seasonality pattern. However, the traffic flow gradually decreases due to *COVID-19* lockdown. As TES was trained in the *offline training* stage using pre-COVID-19 traffic, it still forecasts a higher number of vehicles/hour than the envisioned after the lockdown, thus its high RMSE in Figure 3b. This is not the case for the TCN forecasting approach, which despite the use of *offline training*, adapts its forecasts, capturing the traffic flow decrease experienced due to the lockdown.

V. FORECAST-BASED SCALING FOR V2N SERVICES

Section V-A presents how existing solutions tackle the V2N scaling problem formulated in Section III, and explains in Section V-B the proposed *n-max* scaling algorithm. In the following, Section V-C compares the performance of *n-max* algorithm against existing state-of-the-art solutions.

A. V2N SCALING SOLUTIONS

As mentioned in Section II, C-V2X scaling solutions are typically based on threshold-based mechanisms. These mostly assume that the latency T_0 in Problem III.1 is exceeded when the edge server reaches its maximum load, i.e., when $\rho_t = 1$. But according to our system model (see Section III), it may happen that, at a given time t, the experienced latency exceeds the constraint $T_t > T_0$ with $\rho_t < 1$, as the latency T_t depends on both the current vehicle arrival rate λ_t, and the number of CPUs c_t allocated in the edge server – see (Eq. 1).

To this extent, we define $\overline{\rho}_C(T_0)$ as the maximum load the edge server can handle to meet a T ms latency constraint when it has C CPUs allocated for V2N traffic processing. For example, $\overline{\rho}_2(5\ ms) = 0.2$ means that an edge server with 2 CPUs will meet the 5 ms latency constraint whenever the load is below 0.2. The next list describes how existing V2N scaling solutions can solve Problem III.1:

- ***Threshold-based [28]***: in our system model, the algorithm proposed in n [28] scales up when

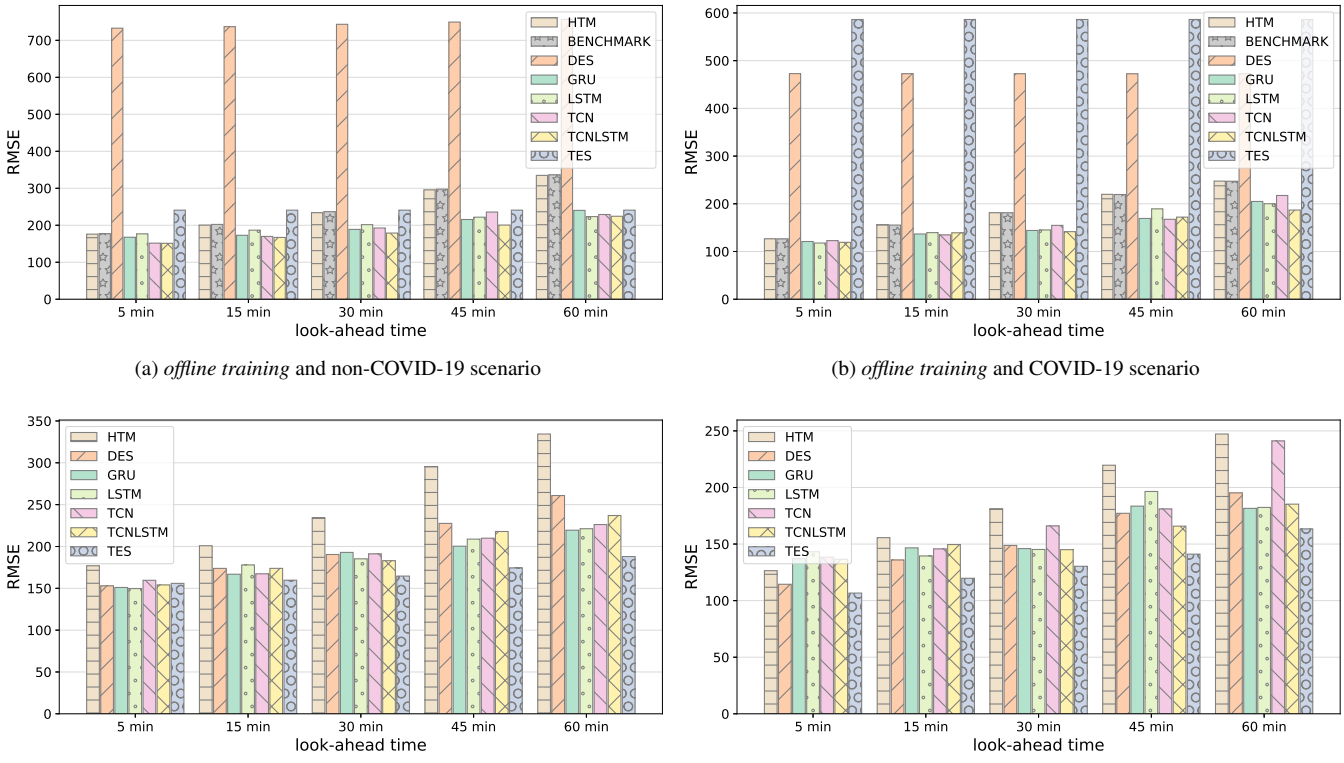
$$\rho_t > \tau \cdot \overline{\rho}_{c_t}(T_0), \quad \tau \in [0, 1] \tag{8}$$

  with τ being a threshold specified by the edge server owner. In other words, if the current load exceeds the maximum load, then the approach in [28] adds an additional CPU. To scale down, we define $\rho_t^* = \frac{\lambda_t}{\mu(c_t - 1)}$ as the load that the edge server would experience without one of its allocated CPUs. Thus, [28] will release a CPU when

$$\rho_t^* < \tau \cdot \overline{\rho}_{c_t - 1}(T_0) \tag{9}$$

  that is, if the load without one CPU is τ times less than then the maximum load that supports a latency of T_0 ms.
- ***Threshold + wait [27]***: to prevent increasing the amount of CPUs upon spurious peaks of road traffic, the approach in [27] allocates another CPU in the edge server

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access



(a) *offline training* and non-COVID-19 scenario

(b) *offline training* and COVID-19 scenario

(c) *online training* and non-COVID-19 scenario

(d) *online training* and COVID-19 scenario

FIGURE 3: Accuracy of Section IV look-ahead forecasts.

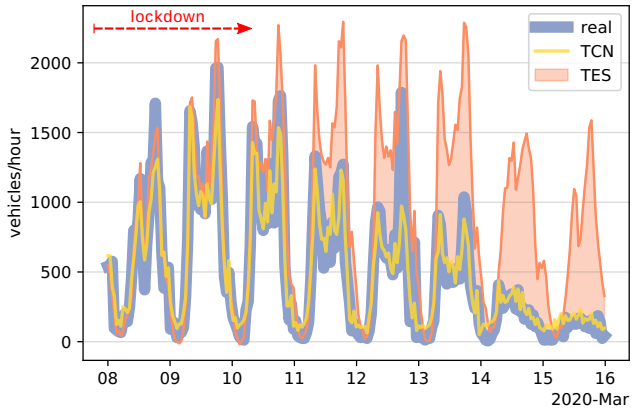


FIGURE 4: TES, TCN forecasts vs. real flow values. 5 min. *look-ahead* in *COVID-19* scenario using *offline training*.

if the threshold τ is exceeded during a waiting period of w time units. That is, one CPE is added when

$$\min\{\rho_{t-w}, \ldots, \rho_t\} > \tau \cdot \overline{\rho}_{c_t}(T_0) \tag{10}$$

Similarly, one CPU is released if

$$\max\{\rho_{t-w}, \ldots, \rho_t\} < \tau \cdot \overline{\rho}_{c_t}(T_0) \tag{11}$$

- ***AutoMEC [26]***: contrary to the former threshold solutions, AutoMEC does not trigger the scaling based on load thresholds, but rather on the predicted increase in the arrival rate. To derive the traffic predictions $\hat{\lambda}_{t+n}$, AutoMEC uses a LSTM neural network. In case the condition

$$\hat{\lambda}_{t+n} > \alpha \cdot \lambda_t \tag{12}$$

is satisfied, AutoMEC will scale. Condition (Eq. 12) uses a factor α that weights the scaling decision based on the forecasting accuracy, namely, $\alpha = \frac{a_r}{a}$ with $a \in [0, 1]$ being the forecasting accuracy of the LSTM prediction, and $a_r \in \mathbb{R}^+$ the relevance given to such prediction. Hence, if (Eq. 12) is satisfied AutoMEC allocates c_{t+1} CPUs. The number of allocated CPUs satisfied

$$\frac{\lambda_t + (\lambda_t - \alpha\hat{\lambda}_{t+n})}{c_{t+1} \cdot \mu} < \overline{\rho}_{c_{t+1}}(T_0) \tag{13}$$

That is, AutoMEC chooses c_{t+1} to accommodate an additional demand of $\lambda_t - \alpha\hat{\lambda}_{t+n}$. Thus it ensures that the load that satisfies the latency constraint T_0 will not be exceeded. Similarly, when the following formula is satisfied the number of allocated CPUs c_{t+1} should also satisfy (Eq. 13).

$$\hat{\lambda}_{t+n} < \alpha \cdot \lambda_t \tag{14}$$

On top of the list above with state-of-the-art V2N scaling techniques, over-provisioning, and average scaling are also considered for comparison latter in Section V-C:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

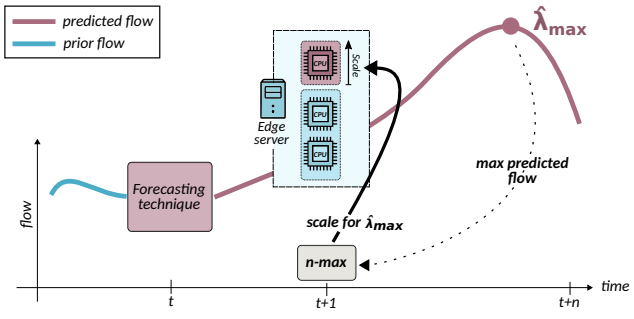


FIGURE 5: Illustration of *n-max* scaling.

- **Over-provisioning**: this solution assumes that the allocated CPUs c is fixed to satisfy the latency constraint T_0 upon a maximum arrival rate λ_{max}

$$\frac{\lambda_{max}}{c \cdot \mu} \leq \overline{\rho}_c(T_0) \tag{15}$$

  where $\lambda_{max} = \max\{\lambda_{t-j}\}_{j=0}^{\infty}$.
- **Average scaling***:* contrary to the prior solution, this one fixes the number of allocated CPUs c to meet the latency constraint for the average arrival rate λ_{avg}

$$\frac{\lambda_{avg}}{c \cdot \mu} \leq \overline{\rho}_c(T_0) \tag{16}$$

  where $\lambda_{avg} = \lim_{N\to\infty} \frac{1}{N} \sum_{j=0}^{N} \lambda_{t-j}$.

B. *N-MAX* SCALING ALGORITHM

This section describes *n-max*, the V2N scaling solution proposed by this paper. The algorithm utilizes the best forecasting algorithm, according to the performance analysis in Section IV-B (TES with online training, as shown in Table 4), to predict the upcoming road traffic for the next n timestamps $\hat{\lambda}_{t+1}, \ldots, \hat{\lambda}_{t+n}$. Based on the prediction, it allocates a sufficient number of CPUs to satisfy the latency requirement T_0. In particular, c_{t+1} is set so that:

$$\max\left\{\frac{\hat{\lambda}_j}{c_{t+1}\mu}\right\}_{j=t+1}^{t+n} \leq \overline{\rho}_{c_{t+1}}(T_0) \tag{17}$$

That is, *n-max* sets the number of CPUs c_{t+1} such that the maximum forecasted load (left term) remains below the maximum load to satisfy the latency constraint T_0 (right term). Prior state-of-the-art scaling solutions only compute the required number of CPUs if the scaling conditions are met (see Section V-A). On the other hand, *n-max* checks c_{t+1} each time it forecasts the incoming demand. The frequency at which *n-max* triggers a forecast is a parameter that the user can decide. It is worth mentioning that upon predictions of future traffic loads, *n-max* allocates enough CPUs to process on time the future peaks. This is due to the maximum considered in (Eq. 17). Overall, *n-max* procedure is summarized as follows:

i) Forecast the traffic n steps ahead $\hat{\lambda}_{t+n}$ using the best forecasting technique in Table 4;

ii) Compute the maximum traffic forecasted for the n steps ahead $\hat{\lambda}_{\max} = \max\{\hat{\lambda}_{t+1}, \ldots, \hat{\lambda}_{t+n}\}$; and

iii) Scale the number of CPUs in the next timestamp c_{t+1} to meet the maximum traffic forecasted $\hat{\lambda}_{\max}$.

Figure 5 illustrates the described steps. At time t *n-max* invokes the best forecasting technique (i.e., TES with online training) and obtains the predicted traffic flow n steps ahead (until $t+n$). Based on the maximum predicted flow $\hat{\lambda}_{\max}$, *n-max* scales up another CPU such that at $t+1$ the edge server can already accomodate a demand $\hat{\lambda}_{\max}$. In other words, *n-max* anticipates the scaling to meet the incoming forecasted peak of demand $\hat{\lambda}_{\max}$.

Algorithm 1 details how *n-max* works. The algorithm has a frequency parameter F that details how often *n-max* is invoked (see line 1). Given that our dataset has a granularity of 5 min., F should satisfy $F \equiv 0 \mod 5$, with F expressed in minutes. If we take $F = 10$, this will result in entering the scaling routine every 10 min. In case we enter in the scaling routine, the first thing to do is to forecast the flow for the n time steps ahead using a forecasting function $f(X_{t,h})$ (e.g., TES with online training), as shown in line 2. Later, we compute what is the maximum forecasted flow $\hat{\lambda}_{\max}$ in line 3.

Once the maximum forecasted flow is computed, *n-max* enters in a loop in line 5, and starts to increase the number of future CPUs c_{t+1} until it ensures that the maximum demand can be accommodated, that is, it keeps increasing the number of CPUs as long as the average latency remains above the target delay T_0. Remember that in Section III we consider the edge server as an $M/M/c$ queue, hence, *n-max* keeps increasing the number of CPUs if the average sojourn time with demand $\hat{\lambda}_{\max}$ stays above T_0 (see line 8). Note that this is equivalent to increasing the number of CPUs until the load remains below $\overline{\rho}_{c_{t+1}}(T_0)$, as stated in (17). Line 6 computes the Earlang-C formula for the maximum demand $\hat{\lambda}_{\max}$, to later compute the average sojourn time and decide if *n-max* keeps increasing the number of CPUs. If *n-max* exits the do-while loop (line 9), that means that it has already

Algorithm 1: *n-max* scaling algorithm

Data: μ, T_0, n, F
1 **for** $t \in \left\{\frac{i \cdot F}{5\ min.} : i \geq 0\right\}$ **do**
2 $\hat{\lambda}_{t+n}, \ldots, \hat{\lambda}_{t+1} = f(X_{t,h})$;
3 $\hat{\lambda}_{max} = \max\left\{\hat{\lambda}_{t+i}\right\}_{i=1}^{n}$;
4 $c_{t+1} = 1$;
5 **do**
6 $P_Q(c_{t+1}, \hat{\lambda}_{max}) = \frac{p_0\left(c_{t+1}\frac{\hat{\lambda}_{max}}{\mu}\right)^{c_{t+1}}}{c_{t+1}!\left(1-\frac{\hat{\lambda}_{max}}{\mu}\right)}$;
7 $c_{t+1} = c_{t+1} + 1$;
8 **while** $\frac{1}{\mu} + \frac{P_Q(c_{t+1}, \hat{\lambda}_{max})}{c_{t+1}\mu - \hat{\lambda}_{max}} > T_0$;
9 scale(c_{t+1});
10 **end**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

TABLE 3: Scaling worst-case run-time complexity

	Threshold-based, Threshold+wait	**AutoMEC**	**Average, over-provisioning, *n-max***
Complexity	$\mathcal{O}\left(\frac{\lambda_{\max} \cdot c_{\max}^2}{\delta}\right)$	$\mathcal{O}\left(\frac{\lambda_{\max} \cdot c_{\max}^3}{\delta}\right)$	$\mathcal{O}(c_{\max}^3)$

increased the number of CPUs enough to meet (on average) the target latency T_0; and that is the number of CPUs c_{t+1} that are required in the scaling.

We now proceed and present the run time complexity analysis of the *n-max* scaling algorithm. To derive the number of operations we resort to the prior summary $i) - iii)$ of the steps that *n-max* makes:

- ***i***) Forecasting the traffic for the next n steps takes as many operations as required by the forecasting technique $f(X_{t,h})$ in Algorithm 1, line 2. In the performance evaluation in Section V-C we use TES for $f(X_{t,h})$, which makes a linear amount of operations on the step size $\mathcal{O}(n)$;
- ***ii***) Computing the maximum traffic forecasted for the n steps ahead takes also a linear amount of operations $\mathcal{O}(n)$; and
- ***iii***) Scaling the number of CPUs is the most complex operation, for it enters the loop to compute the Earlang-C formula $P_Q(c_{t+1}, \hat{\lambda}_{max})$, and check if the average sojourn time is satisfied (line 8). In particular, in Appendix B, we proof that Algorithm 1's loop has a run-time complexity of is $\mathcal{O}(c_{\max}^3)$, for it is dominated by the computation of the Earlang-C formula. With $c_{\max}$ we refer to the maximum number of CPUs that we can scale up in the edge server.

Hence, the *n-max* algorithm is dominated by the scaling loop, and its worst-case run-time complexity is $\mathcal{O}(c_{\max}^3)$. In Appendix B we also proof the run time complexity of the other state-of-the-art algorithms that we introduced in Section V-A. Table 3 summarizes the complexity of both *n-max* and the state-of-the-art scaling algorithms, and shows that *n-max* worst-time complexity is better than AutoMEC (the other forecasting-based scaling solution that we presented in Section V-B). In Table 3, δ represents the numerical precision of the arrival rate λ – see Appendix B. Higher precision is achieved with smaller values of δ, hence, the precision results in an increase in the run-time complexity due to the $\frac{1}{\delta}$ factor in the worst-case complexity in Table 3.

C. FORECAST-BASED SCALING PERFORMANCE

Given the system model of Section III, this Section analyses the performance of the proposed *n-max* algorithm to scale remote driving, cooperative awareness, and hazard warning V2N services.

The algorithm's performance is assessed by means of cost savings and latency violations. Moreover, *n-max* is compared against existing scaling strategies explained in Section V-A. Experiments used the most accurate forecasting technique among the ones evaluated in Section IV-B. Finally, results are

TABLE 4: Best Traffic Flow Forecasting Techniques

Forecasting task		**Most accurate**	
Step-ahead	**Scenario**	**Technique**	**Online**
5 min	*non-COVID-19*	LSTM	✓
	COVID-19	TES	✓
15 min	*non-COVID-19*	TES	✓
	COVID-19	TES	✓
30 min	*non-COVID-19*	TES	✓
	COVID-19	TES	✓
45 min	*non-COVID-19*	TES	✓
	COVID-19	TES	✓
60 min	*non-COVID-19*	TES	✓
	COVID-19	TCNLSTM	✓

derived using *(i)* a real traffic dataset from the city of Torino; and *(ii)* reference service rate values reported by a European Research project, namely 5G-TRANSFORMER.

In particular, the service rate μ is obtained from 5G-TRANSFORMER [39] that reports the results of an Enhanced Vehicular Service (EVS); this is a service that deploys sensing, video streaming, and processing facilities to the edge. The deliverable reports not only the required physical resources to deploy an EVS service, but also the flow of cars used to perform their evaluations. Moreover, it details that an EVS instance, i.e. $c = 1$ in our notation, offers a service rate of $\mu_{EVS} = 208.37$ vehicles/second.

The experiments consist in running the proposed *n-max* scaling algorithm in the *COVID-19* scenario. In particular, *n-max* decides what is the required number of servers c_t to meet the V2N service latency requirement T_0 within the next n minutes. The value of μ is set to be proportional to μ_{EVS} depending on the V2N service, and traffic flow forecasting (Algorithm 1, line 2) is performed using TES with online training, which was the technique that gave the lowest RMSE for n minutes look-ahead predictions (see Table 4).

Figure 6 and Figure 7 compare the performance of the proposed *n-max* scaling algorithm against the existing state-of-the-art solutions presented in Section V-A. Every solution was tested in the COVID-19 scenario, and both AutoMEC and *n-max* performed scaling actions considering forecasts of 30, 45, and 60 minutes ahead. Remote driving, cooperative awareness, and hazard warning were the considered services in the experiments. Each V2N service has different latency requirements T_0 and service rates μ. Namely, *(i)* remote driving has a latency constraint of $T_0 = 5$ ms and the service rate was set to be $\mu = \mu_{EVS}$; *(ii)* cooperative awareness asks for a latency constraint of $T_0 = 100$ ms and we set a service rate of $\mu = \frac{\mu_{EVS}}{20}$; and *(iii)* hazard warning needs latencies below $T_0 = 10$ ms and experiments were executed with a service rate $\mu = \frac{\mu_{EVS}}{2}$.

In the experiments, AutoMEC was executed with $\alpha = 0.8$. This was the value that achieved the best performance by means of cost and delay, given that the accuracy of the offline trained LSTM is $a = 0.36$, $a = 0.37$, and $a = 0.42$ for 30, 45, and 60 minutes forecasts; respectively. While searching for the best α value for AutoMEC, only values of $\alpha^2 < 1$ were considered to prevent AutoMEC from not scaling (see Appendix A for further details).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

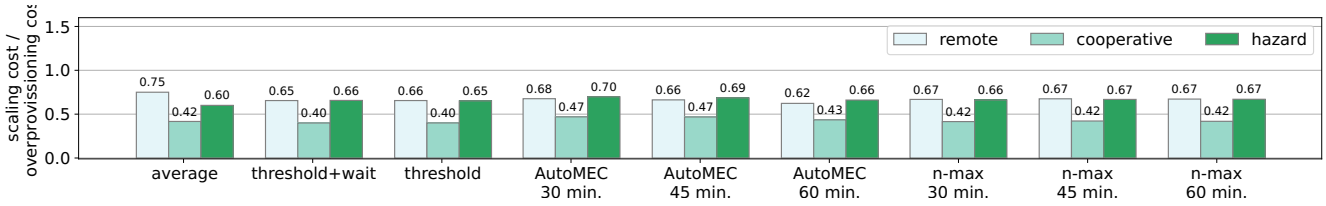


FIGURE 6: Cost savings of SoA and proposed scaling solution (*n-max*).

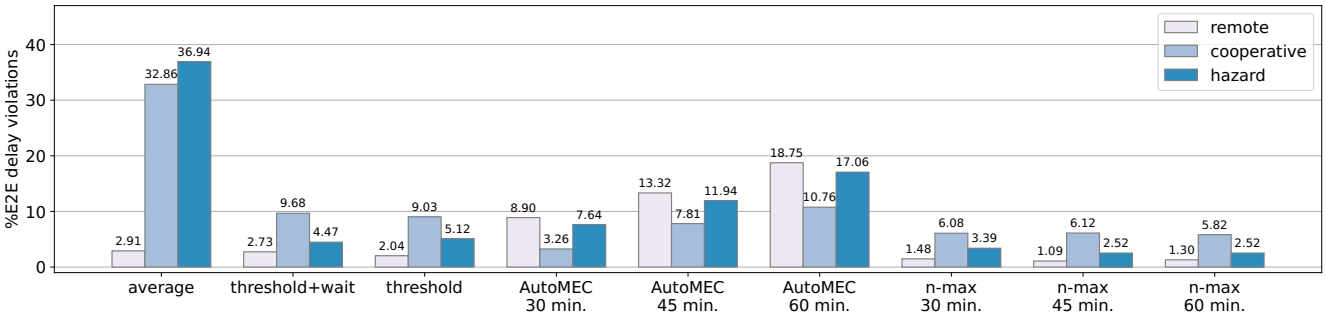


FIGURE 7: Delay violations due to SoA and proposed scaling solution (*n-max*).

Both Figure 6 and Figure 7 are complementary to understand the cost and delay trade-off among the different solutions. In Figure 6, the bars illustrate the cost ration between over-provisioning scaling and each solution. A ratio of 1 would mean that the considered solution (e.g., average scaling) costs as much as over-provisioning CPUs for the V2N service. Figure 7 illustrates the corresponding percentage of delay violations for each V2N service during the COVID-19 scenario.

As expected, Figure 6 shows that every scaling solution reduces the scaling cost compared to over-provisioning. In particular, they lead to costs that are below a 75% of the over-provisioning approach. In the case of cooperative awareness, and hazard warnings, the scaling costs are below a 47% and 69% of the over-provisioning case; respectively. The proposed *n-max* algorithm with 30 min. forecasts is a 5% more expensive that AutoMEC with 60 min. forecasts when scaling remote driving services. It is also a 2% more expensive than the threshold+wait and threshold solutions in the case of cooperative driving , and a 6% more expensive than average scaling in hazard warning services.

However, Figure 7 shows that every *n-max* solution, with either 30-60 min ahead forecasts, has fewer delay violations than all other solutions in remote driving and hazard warning scenarios. In particular, *n-max* with forecasts of 45 min results in only a 1.09% of delay violations in a remote driving service, and just a 2.52% of violations in hazard warning. For the cooperative awareness service, AutoMEC with 30 min forecasts achieves the lowest number of delay violations (just a 3.26%), followed by *n-max* scaling, which leads to 5.82% delay violations. This difference in the number of violations is due to the fact that AutoMEC with 30 min allocates more CPUs for the remote driving service (see in Figure 6 how its cost is higher than *n-max* with 60 min. forecasts).

Figure 8a and Figure 8b give insights on how each scaling solution works in the cooperative awareness scenario. [3] The illustrated time-lapse conveys both the end and beginning of a day in Torino. As shown in between 18:00 and 20:00, the threshold solution incurs in a ping-pong effect due to the oscillation of traffic demand, whilst the threshold+wait solution prevents such effect in the two hours interval. However, the waiting in the latter solution leads to an under-provisioning that causes the violation of the 100 ms delay constraint in between 6:00 to 8:00 of the next day (see Figure 8a). That is, when the day starts and traffic increases, the threshold+wait solution reacts late and does not allocate enough resources for the cooperative awareness demand. Nevertheless, also the threshold-based solution and AutoMEC with 30 min. of forecasts lead to delay violations in the increase of traffic foreseen in 6:00-8:00. It is only the *n-max* algorithm which predicts such demand increase, and preemptively allocates enough CPUs to process on-time V2N service requests.

However, our proposed *n-max* solution also presents drawbacks in the cooperative awareness time-lapse of Figure 8. Contrary to the remote driving and hazard warning V2N services, *n-max* resulted in a resource under-provisioning that lead to the violation of the 100 ms latency constraint of the cooperative awareness service (see Figure 8b around 18:00, 20:00, and the start of 1st March). This explains why *n-max* with 60 min. forecasts save more cost in the scaling process than AutoMEC with 30 min. forecasts (see Figure 6), since *n-max* is more prone than AutoMEC to under-provisioning in such a scenario. As a consequence, in Figure 7 *n-max* with 60 min. forecasts incur in a 2.56% of additional latency violations.

[3]We choose cooperative awareness because the scaling requires more CPUs due to its low service rate μ, thus, it evidences better the algorithm differences.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

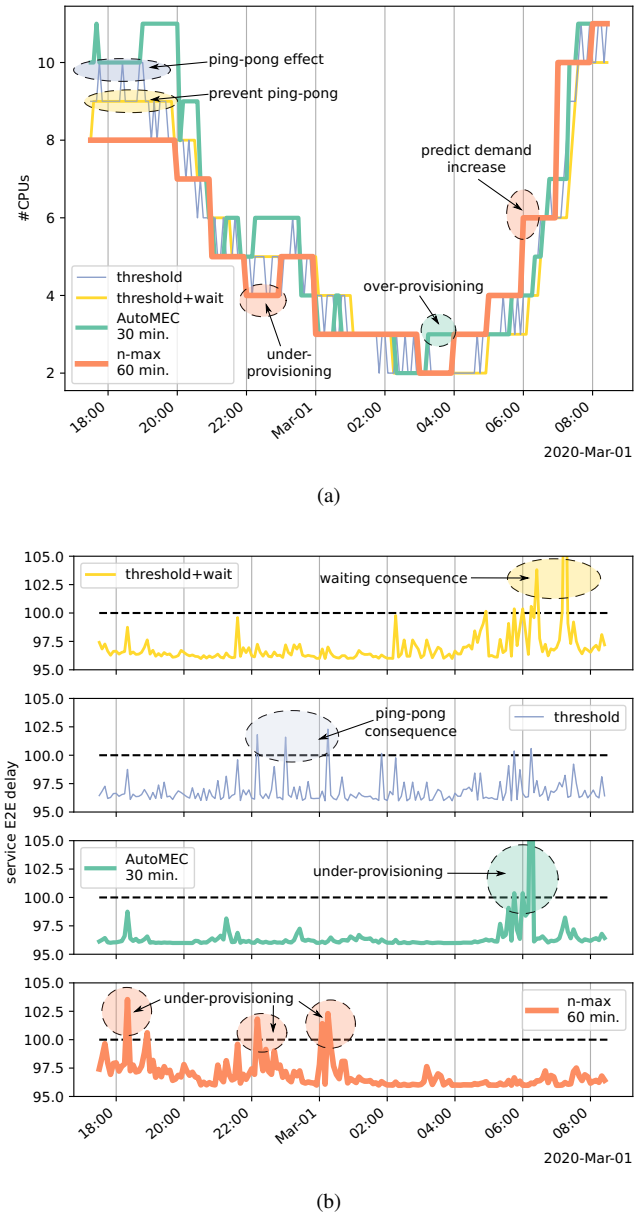


FIGURE 8: Impact of cooperative awareness scaling on (a) allocated CPUs; and (b) latency violations. TES with online training was used for *n-max* with 60 min. predictions.

In summary, experiments show that *n-max* with online TES forecasting prevents the ping-pong scaling and awaiting artifacts foresaw in another state-of-the-art solutions (see Figure 8). Hence, *n-max* with TES online forecasting reduces the E2E delay violations (see Figure 7) in remote driving by more than a half (from 2.04% in threshold-based scaling, down to a 1.09% in *n-max* with $n = 45$ min.), and by almost a half in hazard warning use cases (from 4.47% in the threshold+wait solution, down to 2.52% in *n-max* with $n = 45, 60$ min.).

VI. CONCLUSIONS AND FUTURE WORK

This paper provides an extensive analysis of state-of-the-art techniques to forecast the road traffic for the city of Torino, either based on traditional time-series methods or on ML-based techniques. The performed analysis compares each forecasting technique's RMSE by considering *(i)* forecasting intervals from 5 to 60 minutes; *(ii) offline/online training*; and *(iii)* COVID-19 lockdown. Results show that under *offline training*, ML-based techniques outperform traditional time-series methods, especially during the COVID-19 lockdown, as they adapted to the Torino traffic drop better. With online training, time-series techniques achieve results better or as good as the analyzed ML-based techniques.

Furthermore, we introduce a V2N scaling algorithm *(n-max)*, which leverages on the most accurate forecasting technique, and evaluate its performance via simulation.

Results show that *n-max* outperformed existing solutions to scale remote driving and hazard warning services, resulting in the lowest E2E delay violations. However, when it comes to E2E delay violations in cooperative awareness services, AutoMEC is able to perform better due to over-provisioning.

A first direction to extend this work is to consider other time-series forecasting solutions (as Prophet) to boost the scaling performance of *n-max*, and to find techniques that can incorporate information neighboring road probes, such as spatial analysis techniques. Furthermore, the applicability of the presented techniques to different scenarios is also envisioned as a next step. The use of different datasets, including operator records with respect to the base stations used by mobile phones to access the Internet, is also going to be taken into consideration depending on the availability of datasets. In such a scenario, forecasting the user density distribution along time would enable better decisions regarding the edge server placement and service migrations.

Similar to the adopted scaling strategy of this work, enhancing orchestration algorithms with forecasting information would contribute to smarter orchestration and resource control. The resulting decisions would be impacted in terms of improved quality and accuracy. Optimized deployment, enhanced management and control of elastic network slices that support dynamic demands and their respective SLAs, improved resource arbitration and allocation, or maximized service request admission, are some examples where forecasting information can impact the decisions.

The aforementioned mechanisms are going to be developed and leveraged in selected use cases in the scope of the 5Growth project, which comprises Industry 4.0, transportation, and energy scenarios. They will be integrated to support full automation and SLA control for elastic network services life-cycle management. Hence, it would be worth studying the probability of forecasting less demand than what is required by each use case, i.e., $\mathbb{P}(\hat{F} < F)$; so as to perform preemptive actions under high probabilities of forecasting error. Such a calculus deserves a detailed analysis on how to compute max-statistics for correlated random variables (e.g.,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

speed and traffic flow) [40].

.

APPENDIX A AUTOMEC α CONSTRAINT

The AutoMEC algorithm [26] was considered for comparison in this paper. Its scaling condition (Eq. 12) uses a parameter $\alpha = \frac{a_r}{a}$ to weight the scaling decision based on the LSTM forecasting accuracy a, and the relevance a_r given to the forecasting. Given the accuracy a of the LSTM forecasting, [26] does not provide insights on how to select a_r. This appendix shows that a_r must be selected to satisfy $\alpha^2 < 1$, otherwise, AutoMEC never increases the number of allocated CPUs. Thus, the election of a value of $\alpha^2 < 1$ in Section V-C performance evaluation.

Lemma A.1. *If $\alpha^2 \geq 1$, AutoMEC never increases the number of CPUs.*

Proof. According to [26, Algorithm 1], AutoMEC scales the number of CPUs when the forecasted future demand $\hat{\lambda}_{t+n}$ satisfies

$$\hat{\lambda}_{t+n} > \alpha \cdot \lambda_t \tag{18}$$

In particular the number of additional CPUs is expressed as

$$c^+_{t+1} = \left\lceil \frac{\lambda_t - \alpha \cdot \hat{\lambda}_{t+n}}{c_t \cdot \mu} \right\rceil \tag{19}$$

Given that condition (18) is satisfied, we have $c^+_{t+1} \leq \lceil \frac{\lambda_t(1-\alpha^2)}{c_t \cdot \mu} \rceil$. If $\alpha^2 \geq 1$, this means that $c^+_{t+1} \leq 0$, and AutoMEC will never increase the number of CPUs. □

APPENDIX B ALGORITHMS RUN-TIME COMPLEXITY

Here we analyze what is the worst-case run-time complexity of the algorithms presented in Section V-A. All algorithms are based on the maximum load accepted to meet the target delay T_0, i.e., all algorithms are formulated based on $\overline{\rho}_c(T_0)$. Hence, we should look at it to derive the run-time complexity of our algorithms.

We can express the average sojourn time as a function

$$T = g(\lambda, \mu, c) = \frac{1}{\mu} + \frac{1}{c\mu - \lambda} \cdot \frac{p^c_0(c\rho)}{c!(1-\rho)} \tag{20}$$

with $\rho = \frac{\lambda}{c\mu}$. Note that (20) corresponds to the expression given in (Eq. 1). And we see that the maximum load that meets a target delay T_0 is precisely the inverse of the average sojourn time, i.e., $\overline{\rho}_c(T_0) = g^{-1}(\rho)$. However, we should express $g(\cdot)$ in terms of ρ, and even if we did that, still $g(\cdot)$ would not be an invertible function. Rather than computing an approximation of the inverse function, we fix some input parameters, and iterate over a single input parameter – as λ or c – until $g(\cdot) = T_0$.

So, lets check the complexity of evaluating $g(\cdot)$. If the reader checks (Eq. 20), the dominating term by means of operations is $p^c_0(c\rho)$, whose expression is given in (Eq. 3). Thus, we can state:

Lemma B.1. *Given that the maximum number of CPUs in an Edge server is $c_{\max}$, the worst-case run-time complexity of $p_0(c\rho)$ is $\mathcal{O}(c^2_{\max})$.*

Proof. Following (3), the most dominating term is the summation $\sum_{n=0}^{c_{\max}-1} \frac{(c\rho)^n}{n!}$, which unrolls as:

$$\frac{0}{0!} + \frac{c\rho}{1!} + \frac{(c\rho)\cdot(c\rho)}{2\cdot 1} + \ldots + \frac{\overbrace{(c\rho)\cdot\ldots\cdot(c\rho)}^{2\cdot(c_{\max}-2)\text{ multiplications}}}{\underbrace{(c_{\max}-1)\cdot(c_{\max}-2)\cdot\ldots\cdot 1}_{c_{\max}-2\text{ multiplications}}} \tag{21}$$

and the number of multiplications/divisions performed is $\sum_{n=0}^{c_{\max}-1} 3n+1$, which is equivalent to $\frac{1}{2}(3c_{\max}-1)c_{\max} = \mathcal{O}(c^2_{\max})$. Hence, the computation of $p_0(c\rho)$ has worst-case complexity $\mathcal{O}(c^2_{\max})$. □

Equipped with the above lemma, we know the complexity of evaluating $g(\cdot)$, which is what we are looking for:

Corollary B.2. *Computing the average sojourn time of an $M/M/c$ system has worst-case run-time complexity $\mathcal{O}(c^2_{\max})$.*

Proof. If we check (Eq. 20), the dominating term by means of operations (multiplications) is the computation of $p_0(c\rho)$; which has complexity $\mathcal{O}(c^2_{\max})$ based on what we have just shown in Lemma B.1. Hence, the average sojourn time calculus $g(\lambda, \mu, c)$ has worst-case complexity $\mathcal{O}(c^2_{\max})$. □

Now that we know how much it takes to evaluate $g(\cdot)$, we can derive the complexity of $\overline{\rho}_c(T_0)$, i.e., the maximum load that c CPUs stand to dispatch requests below a time T_0, on average. As aforementioned, $\overline{\rho}_c(T_0) = g^{-1}(\cdot)$, but g is not an invertible function, so we have to fix two input parameters of $g(\lambda, \mu, c)$ and iterate over the other depending on what we want:

- ***Arrival rate*** λ: in this case we fix μ_0, c_0 and iterate taking steps of size δ until we satisfy $g(\lambda_{\min} + i\delta, \mu_0, c_0) < T_0$, with $i \in \{0, 1, \ldots, \frac{\lambda_{\max}}{\delta}\}$. In other words, we are looking for the maximum load that can be satisfied on time, in particular we look for the number of steps $i_{\max} = \max_i\{g(\lambda_{\min} + i\delta, \mu_0, c_0) < T_0\}$, such that $\overline{\rho}(T_0) \simeq \frac{\lambda_{\min} + i_{\max}\delta}{c_0\mu_0}$. Since we need to iterate over different values of i and check if $g(\cdot) < T_0$ each time, the worst-case run-time complexity is $\mathcal{O}(\frac{\lambda_{\max}c^2_{\max}}{\delta})$; or
- ***number of CPUs*** c: in this case we fix λ_0, μ_0 and evaluate $g(\lambda_0, \mu_0, c) < T_0$ with $c = 0, 1, \ldots, c_{\max}$, And increase c until $\overline{\rho}(T_0) \simeq \frac{\lambda_0}{c\mu_0}$. Then, the worst-case run-time complexity will be $\mathcal{O}(c^3_{\max})$, for we have to evaluate $g(\cdot)$ (with complexity $\mathcal{O}(c^2_{\max})$ according to Corollary B.2) a maximum of $c_{\max}$ times.

Taking these two cases into account, we can derive the worst-case run-time complexity of the algorithms in Section V-A; depending on whether the value they look for, the arrival rate, or the number of CPUs:

- ***Threshold solutions***: both threshold-based and threshold+wait fix the number of CPUs to $c = c_t$, and compute the value of $\overline{\rho}_{c_t}(T_0)$. In other words, both look for the maximum arrival rate that can be processed below T_0

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

seconds in (Eq. 8) and (Eq. 10), respectively; and scale up another CPU (same for scaling down). In both cases, the dominating term by means of complexity is the computation of $\overline{\rho}_{c_t}(T_0)$, which is $\mathcal{O}(\frac{\lambda_{\max} c_{\max}^2}{\delta})$. Thus, the run-time complexity is reported in Table 3;

- ***AutoMEC***: this solution performs a forecasting of the future load $\hat{\lambda}_{t+n}$, and checks in (Eq. 12) if it has to scale up resources. The complexity[4] of computating a forward pass in a LSTM network is $\mathcal{O}(h \cdot m)$, with h the history size (12 samples related to 60 min. in our case), and m the number of neurons in a hidden layer (100 in our experiments). Given the forecast, which is not the dominating term, AutoMEC decides the number of CPUs to set in (Eq. 13). In particular, it iterates over $c = 0, 1, \ldots, c_{\max}$, and for each value of c it looks for the maximum arrival rate it can process below T_0 seconds. In other words, given c it looks for λ that satisfies $g(\lambda, \mu_0, c) < T_0$. As aforementioned, this has a run-time complexity of $\mathcal{O}(\frac{\lambda_{\max} c_{\max}^2}{\delta})$, and such operation is performed $c_{\max}$ times. Thus, the worst-case run-time complexity of AutoMEC is $\mathcal{O}(\frac{\lambda_{\max} c_{\max}^3}{\delta})$, as shown in Table 3; and
- ***average, over-provisioning, n-max***: both the average, and over-provisioning solutions fix λ to λ_{avg} or $\lambda_{\max}$, respectively; and compute the value of $\overline{\rho}_c(T_0)$. This means that they iterate over c until $g(\lambda_{avg,\max}, \mu_0, c) < T_0$ is satisfied. As shown in the second item in the prior list, this implies that both solutions have a worst-case complexity of $\mathcal{O}(c_{\max}^3)$. Also the *n-max* algorithm has the complexity $\mathcal{O}(c_{\max}^3)$, for the loop in Algorithm 1 iterates increasing the number of CPUs up to $c_{\max}$, and computes $g(\hat{\lambda}_{\max}, \mu, c)$ (with complexity $\mathcal{O}(c_{\max}^2)$ according to Lemma B.1) in every comparison at line 8 – with μ a fixed value given in the input.

[4] An LSTM forward pass imply matrix-vector multiplications – see [36].

REFERENCES

[1] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network Slicing in 5G: Survey and Challenges," IEEE Communications Magazine, vol. 55, no. 5, pp. 94–100, 2017.

[2] R. G. Brown, Smoothing, forecasting and prediction of discrete time series. Prentice Hall, 1963.

[3] Y.-S. Lee and L.-I. Tong, "Forecasting time series using a methodology based on autoregressive integrated moving average and genetic programming," Knowledge-Based Systems, vol. 24, no. 1, pp. 66–72, 2011.

[4] A. Marotta, D. Cassioli, K. Kondepu, C. Antonelli, and L. Valcarenghi, "Exploiting flexible functional split in converged software defined access networks," J. Opt. Commun. Netw., vol. 11, no. 11, pp. 536–546, 11 2019.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput., vol. 9, no. 8, p. 1735–1780, 11 1997.

[6] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "Statistical and machine learning forecasting methods: Concerns and ways forward," PLOS ONE, vol. 13, no. 3, pp. 1–26, 3 2018.

[7] V. R. Chintapalli, K. Kondepu, A. Sgambelluri, A. Franklin, B. R. Tamma, P. Castoldi, and L. Valcarenghi, "Orchestrating edge- and cloud-based predictive analytics services," in 2020 European Conference on Networks and Communications (EuCNC), 2020, pp. 1–6.

[8] M. Lippi, M. Bertini, and P. Frasconi, "Short-Term Traffic Flow Forecasting: An Experimental Comparison of Time-Series Analysis and Supervised Learning," IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 2, pp. 871–882, 2013.

[9] Y. Lv, Y. Duan, W. Kang, Z. Li, and F. Wang, "Traffic Flow Prediction With Big Data: A Deep Learning Approach," IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 2, pp. 865–873, 2015.

[10] F. Kong, J. Li, B. Jiang, and H. Song, "Short-term traffic flow prediction in smart multimedia system for Internet of Vehicles based on deep belief network," Future Generation Computer Systems, vol. 93, pp. 460 – 472, 2019.

[11] M. Aqib, R. Mehmood, A. Albeshri, and A. Alzahrani, "Disaster management in smart cities by forecasting traffic plan using deep learning and gpus," in International Conference on Smart Cities, Infrastructure, Technologies and Applications. Springer, 2017, pp. 139–154.

[12] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu, "LSTM network: a deep learning approach for short-term traffic forecast," IET Intelligent Transport Systems, vol. 11, no. 2, pp. 68–75, 2017.

[13] B. Yang, S. Sun, J. Li, X. Lin, and Y. Tian, "Traffic flow prediction using LSTM with feature enhancement," Neurocomputing, vol. 332, pp. 320 – 327, 2019.

[14] S. Goudarzi, M. N. Kama, M. H. Anisi, S. A. Soleymani, and F. Doctor, "Self-Organizing Traffic Flow Prediction with an Optimized Deep Belief Network for Internet of Vehicles," Sensors, vol. 18, no. 10, 2018.

[15] H. Li, "Research on prediction of traffic flow based on dynamic fuzzy neural networks," Neural Computing and Applications, vol. 27, no. 7, pp. 1969–1980, 2016.

[16] R. Fu, Z. Zhang, and L. Li, "Using LSTM and GRU neural network methods for traffic flow prediction," in 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), 2016, pp. 324–328.

[17] ETSI GS ZSM 001, "Zero-touch network and Service Management (ZSM); Requirements based on documented scenarios," V1.1.1, 2019.

[18] D. M. Gutierrez-Estevez, M. Gramaglia, A. D. Domenico, G. Dandachi, S. Khatibi, D. Tsolkas, I. Balan, A. Garcia-Saavedra, U. Elzur, and Y. Wang, "Artificial Intelligence for Elastic Management and Orchestration of 5G Networks," IEEE Wireless Communications, vol. 26, no. 5, pp. 134–141, 2019.

[19] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization," in IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, 2017, pp. 1–9.

[20] S. Xiao and W. Chen, "Dynamic Allocation of 5G Transport Network Slice Bandwidth Based on LSTM Traffic Prediction," in 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), 2018, pp. 735–739.

[21] I. Alawe, A. Ksentini, Y. Hadjadj-Aoul, and P. Bertin, "Improving Traffic Forecasting for 5G Core Network Scalability: A Machine Learning Approach," IEEE Network, vol. 32, no. 6, pp. 42–49, 2018.

[22] I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, P. Bertin, C. Viho, and D. Darche, "Smart Scaling of the 5G Core Network: An RNN-Based Approach," in 2018 IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–6.

[23] I. Alawe, Y. Hadjadj-Aoul, A. Ksentinit, P. Bertin, C. Viho, and D. Darche, "An Efficient and Lightweight Load Forecasting for Proactive Scaling in 5G Mobile Networks," in 2018 IEEE Conference on Standards for Communications and Networking (CSCN), 2018, pp. 1–6.

[24] A. Okic, L. Zanzi, V. Sciancalepore, A. Redondi, and X. Costa-Pérez, "π-ROAD: a Learn-as-You-Go Framework for On-Demand Emergency Slices in V2X Scenarios," in IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, 2021, pp. 1–10.

[25] W. Wang, Y. Huang, Y. Wang, and L. Wang, "Generalized Autoencoder: A Neural Network Framework for Dimensionality Reduction," in 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2014, pp. 496–503.

[26] U. Fattore, M. Liebsch, B. Brik, and A. Ksentini, "AutoMEC: LSTM-Based User Mobility Prediction for Service Management in Distributed MEC Resources," in Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, ser. MSWiM '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 155–159. [Online]. Available: https://doi.org/10.1145/3416010.3423246

[27] J. Baranda, G. Avino, J. Mangues-Bafalluy, L. Vettori, R. Martínez, C. F. Chiasserini, C. Casetti, P. Bande, M. Giordanino, and M. Zanzola, "Automated deployment and scaling of automotive safety services in 5G-Transformer," in 2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), 2019, pp. 1–2.

[28] I. Sarrigiannis, L. M. Contreras, K. Ramantas, A. Antonopoulos, and C. Verikoukis, "Fog-Enabled Scalable C-V2X Architecture for Distributed 5G and Beyond Applications," IEEE Network, vol. 34, no. 5, pp. 120–126, 2020.
[29] L. Cominardi, L. M. Contreras, C. J. Bernardos, and I. Berberana, "Understanding qos applicability in 5g transport networks," in 2018 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), 2018, pp. 1–5.
[30] ITU-T, "Consideration on 5G transport network reference architecture and bandwidth requirements," International Telecommunication Union - Telecommunication Standardization Sector (ITU-T), Study Group 15 Contribution 0462, 2 2018.
[31] M. H. C. Garcia, A. Molina-Galan, M. Boban, J. Gozalvez, B. Coll-Perales, T. Şahin, and A. Kousaridas, "A tutorial on 5g nr v2x communications," IEEE Communications Surveys Tutorials, pp. 1–1, 2021.
[32] 3GPP, "Release description; Release 15," 3rd Generation Partnership Project (3GPP), Technical Report (TR) 21.915 v15.0.0, 10 2019.
[33] Leonard Kleinrock, Queueing Systems, Volume 2: Computer Applications. Wiley, 1976.
[34] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," Neurocomputing, vol. 262, pp. 134 – 147, 2017, online Real-Time Learning Strategies for Data Streams.
[35] Y. Chauvin and D. E. Rumelhart, Backpropagation: theory, architectures, and applications. Psychology press, 2013.
[36] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.
[37] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 4580–4584.
[38] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks," in Proceedings of the Second International Conference on Learning Representations (ICLR 2014), 2014.
[39] 5G-TRANSFORMER, "5G-TRANSFORMER Report on trials results," European Comission, Tech. Rep. D5.4, 11 2019, Online available: http://5g-transformer.eu/wp-content/uploads/2019/11/D5.4-5G-TRANSFORMER_Report_on_trials_results.pdf.
[40] S. N. Majumdar and A. Pal, "Extreme value statistics of correlated random variables," arXiv preprint arXiv:1406.6768, 2014.

JORGE MARTÍN PÉREZ obtained a B.Sc in mathematics, and a B.Sc in computer science, both at Universidad Autónoma de Madrid (UAM) in 2016. He obtained his M.Sc. and Ph.D in Telematics from Universidad Carlos III de Madrid (UC3M) in 2017 and 2021, respectively. His research focuses in optimal resource allocation in networks, and since 2016 he participates in EU funded research projects in UC3M Telematics department.

KOTESWARARAO KONDEPU is an Assistant Professor at India Institute of Technology Dharwad, Dharwad, India. He obtained his Ph.D. degree in Computer Science and Engineering from Institute for Advanced Studies Lucca (IMT), Italy in July 2012. His research interests are 5G, optical networks design, energy-efficient schemes in communication networks, and sparse sensor networks.

DANNY DE VLEESCHAUWER obtained an MSc. in Electrical Engineering and the Ph.D. degree in applied sciences form the Ghent University, Belgium, in 1985 and 1993 respectively. Currently, he is a DMTS in the access network control department of Nokia Bell Labs in Antwerp, Belgium. Prior to joining Nokia, he was a researcher at Ghent University. His early work was on image processing and on the application of queuing theory in packet-based networks. His current research focus is on the distributed control of applications over packet-based networks.

VENKATARAMI REDDY received his B.Tech. degree in Computer Science and Engineering from Jawaharlal Nehru Technological University Hyderabad, India, in 2008 and the M.Tech. degree in Computer Science and Engineering (Information Security) from the National Institute of Technology Calicut, India, in 2010. He is currently working toward his PhD degree in Computer Science and Engineering at the Indian Institute of Technology Hyderabad (IITH), India. His research interests include 5G, Network Function Virtualization (NFV), Software Defined Networking, and AI in Mobile Networks.

CARLOS GUIMARÃES (M.Sc.'2011, Ph.D.'2019) is a Postdoctoral Researcher at Universidad Carlos III de Madrid (UC3M), where he is pursuing research activities in 5G technologies and main enablers as well as in the application of AI/ML into computer networks.

ANDREA SGAMBELLURI received the master's degree in telecommunications engineering from the University of Pisa, Italy, in 2007, and the Ph.D. degree from the Scuola Superiore Sant'Anna, Pisa, in 2015. In 2016, he was a Postdoctoral Researcher with the KTH Royal Institute of Technology, Onlab, Sweden. He is currently an Assistant Professor with the Scuola Superiore Sant'Anna. His research interests include optical networks control plane, including software defined networking (SDN) protocol extensions, network reliability, segment routing application, and multi-domain orchestration. In March 2015, he won the grand prize at 2015 OFC Corning Outstanding Student Paper Competition.(Based on document published on 19 October 2020).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3142346, IEEE Access

LUCA VALCARENGHI is an Associate Professor at the Scuola Superiore Sant'Anna of Pisa, Italy, since 2014. He published almost three hundred papers (source Google Scholar, May 2020) in International Journals and Conference Proceedings. Dr. Valcarenghi received a Fulbright Research Scholar Fellowship in 2009 and a JSPS "Invitation Fellowship Program for Research in Japan (Long Term)" in 2013. His main research interests are optical networks design, analysis, and optimization; communication networks reliability; energy efficiency in communications networks; optical access networks; zero touch network and service management; experiential networked intelligence; 5G technologies and beyond.

CHRYSA PAPAGIANNI is an assistant professor at the Informatics Institute of the University of Amsterdam. She is part of the Multiscale Networked Systems group that focuses its research on network programmability and data-centric automation. Prior to joining UvA she was a network research engineer for Bell Labs Antwerp, as part of the end-to-end Network Service Automation lab. From 2016 to 2018 she was a Research Scientist for the Institute for Systems Research, at the University of Maryland in the United States. Her research interests lie primarily in the area of programmable networks with emphasis on network optimization and the use of machine learning in networking. She has participated in various EU FIRE and 5G-PPP projects, such as Fed4FIRE+, OpenLab, NOVI and 5Growth working on issues related to network slicing.

CARLOS J. BERNARDOS received his PhD in Telematics from UC3M (Spain), where he works as an associate professor. His current research interests are network virtualization and wireless networks. He is an active contributor to the IETF.