Smart scaling of the 5G core network: an RNN-based approach

Imad ALAWE*, Yassine HADJADJ-AOUL*, Adlen KSENTINI†,
Philippe BERTIN*, Cesar VIHO‡, and Davy DARCHE§
*Firstname.Lastname@b-com.com, IRT b<>com, Rennes, France
†adlen.ksentini@eurecom.fr, EURECOM, Sophia-Antipolis, France
‡Cesar.Viho@irisa.fr, IRISA, Rennes, France
§davy.darche@tdf.fr, TDF, Paris, France

***Abstract*—The upcoming mobile core network, which will be based on Virtual Network Functions (VNF), will face an increase of data traffic on both data and control planes. This is due to the increase of the number of connected devices and the newly 5G supported-services like IoT, Connected Health Care etc. Therefore dynamic and accurate scalability techniques should be envisioned in order to answer the needs, in term of resource provisioning, without degrading the Quality Of Service (QoS) already offered by hardware based core networks. Although provisioning new resources is easier as it is a matter of software deployment, the strategy to use (when to scale and how much to scale) remains complex. In this paper we propose scaling techniques based on neural networks to forecast the upcoming load. Hence scheduling the resource provisioning in a manner that all the needed resources will be deployed and active when the load increases. In the same way, it will scale-in the unneeded resources when the traffic load decreases. The proposal is tested via discrete event simulations using a traffic load dataset provided by a Network Operator. The results show clearly the robustness of our proposal comparing to a threshold-based scaling technique.**

***Keywords*—5G, Scaling, Load Balancing, Mobile Core Network, Prediction, Neural Network**

I. Introduction

With the upcoming generation of mobile network, namely 5G, several new services and markets are expected to emerge (i.e. IOT, Connected Health Care etc.). These new services will lead to an explosion of the number of connected devices, of different types and categories (i.e. watches, sensors, robots etc.). It could reach 28 billions of devices connected to 5G by 2021, in front of 17 billion in 2016 [1].

The increase of the number of User Equipments (UE) will be accompanied by an increase of the traffic over both the data and the control planes of the mobile network, and over both sides: the Radio Access Network (RAN) and the Core Network (CN). Especially, as demonstrated in [2], the Access and Mobility Function (AMF), the new generation component of the Mobility Management Entity (MME), is likely to be congested since it is the single point of access for the control plane in the core network. As shown, the overhead causes an increase in latency for the Network Access Stratum (NAS) procedures, up to some requests' rejects when certain load thresholds are exceeded.

Following the case listed above, the requirement of low latency for 5G networks is far from being respected, and solutions need to be found. In this way, the 3GPP has rethought the Core network architecture, which aims at being more flexible and scalable. The new architecture, namely New-Generation Core (NGC) architecture [3], addresses scalability and flexibility by introducing more modular Network Functions (NF) to compose the control plane service, which could also relies on network virtualization via Network Function Virtualization (NFV).

The NGC is a great step further as it splits the monolithic component MME of the Evolved Packet Core (EPC) (4G core network) into the AMF, the Session Management Function (SMF) and the Unified Data Management (UDM). Indeed, this split offers more flexible opportunities but it doesn't tackle yet the scalability strategies and techniques. Therefore, the main aim of this paper is to tackle the scalability issue in the NGC environment, with a particular focus on the AMF. As the components of the NGC are now modular functions that benefit from NFV, deploying or shutting off the instances of the concerned function is more easier; but deciding when to do it, is still a complex process. Therefore, there is a need to define when to scale and the techniques that best fit for mobile networking.

In this paper, we propose different new proactive scaling techniques and strategies. These techniques are based on machine learning and neural networks and adapted for mobile network. As for the strategies, it allows: (i) to control the load of each AMF in order to decrease the latency for a given procedure, (ii) to dynamically deploy new instances (Scale out) or shut off non used instances (Scale in) depending on the traffic load in the mobile network and in a proactive way. The strategies and techniques are tested and proved using discrete event simulations and based on real mobile load data. Also, the proposed proactive strategies and techniques are compared to a reactive scaling technique, showing the benefits of scaling instances in a proactive manner.

The remainder of this paper is organized as follows. Section II lists and analyses related work on scalability solutions for VNFs and more precisely for mobile core network components. Section III presents the assumed architecture based on NGC and NFV. Section IV describes the proposed solution process from data management to neural network testing. Model evaluation and results analysis are presented in Section

V. Finally Section VI concludes this paper and introduces our future work.

II. Related Work

As the components of the NGC are run as VNFs (using Virutual Machine - VM or containers), the scalability issue is now more a VM (container) deployment issue than a hardware deployment issue. It is crucial to decide accurately when to scale and how many instances to deploy or to shut off. Many solutions are proposed in the literature for VNF scaling.

In [4], [5] and [6], authors have proposed scalability based on thresholds. In this scaling categories usually two thresholds are defined: one for scaling in (SIT) and one for scaling out (SOT). When the load exceeds the SOT, a scale out process is launched. On the other side, when the load goes beneath the SIT, a scale in process is launched. This technique allows to scale the system reactively to the evolution of the system load. However, this technique may leads to oscillations, if the system load goes ahead and then beneath the thresholds. These oscillations will affect the overall system performance as they trigger much physical resources provisioning or releasing. Also, it is worth mentioning that many iterations may be needed in order to get the exact number of needed instances. In addition, reactive techniques act after a load increase or decrease. Thus the time elapsed between the scaling decision and the scaling process may lead to a performance deterioration.

Other technique of VNF scaling is adaptive techniques. In [7], the authors propose a mechanism that combines Q-Learning with Gaussian Processes-based system models, which allows to adapt to dynamic environments and improve the scaling policy before taking any action. Indeed, this proposal, as the authors assume, reacts better than static threshold. However, it remains a reactive solution, and hence inherits the same weaknesses. Also, it should be noted that using Reinforcement learning, may take a considerable time before staring to have the correct decisions. Indeed, this is not acceptable in systems like 5G, due to the time constraint of certain type of applications.

Some papers in the literature have opted for proactive solutions like in [8]. Authors propose a solution based on time series model in order to forecast CPU usage from a historical dataset, and hence schedule resource provisioning pro-actively. This technique is interesting for its pro-activity, but it does not offer adaptation in case of traffic pattern evolution and is not compliant with mobile networks' requirements. Besides, the proposed solution considers only system level information, like CPU or memory usage, but for mobile networking in 5G, service level information should be taken into consideration aiming at reflecting better the global load of the core network.

III. Architecture

In this paper, we consider using the New Generation Core Network (NGC). This architecture is the new 5G standard prepared by the 3GPP group. The architecture of the NGC is illustrated in Figure 1. Our work focuses on the Access

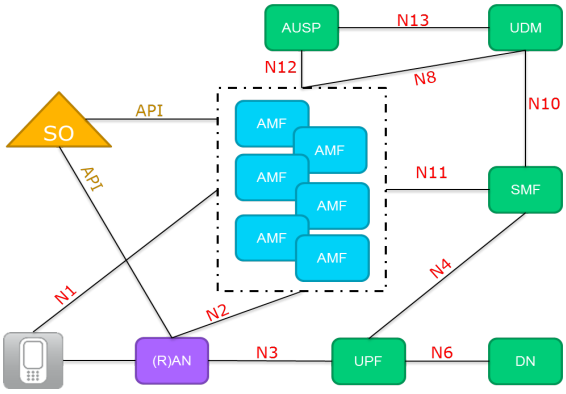

Fig. 1: Architecture proposal for 5G

and Mobility Function (AMF) and the the Unified Data Management (UDM), known as the MME in the EPC.

In legacy 4G core network, the procedures of an attached UE on a certain MME should be processed only by this same MME. Thus, scaling-in an MME means transferring all the users' context to another MME before shutting off the first one. On the other hand, when scaling out, two solutions are possible: (i) the new MME handles new UEs and hence the deployed MMEs continue to serve the already connected UEs, (ii) some migration procedures should be launched, for load balancing purpose, to move some of the already attached UEs to the recently deployed MME. This fact made scalability a complex issue and shows that the EPC architecture is inflexible and especially the MME as it is not stateless.

Therefore, the 3GPP group decided to split the MME component into AMF and UDM, allowing to separates the User Context from the Workers. In other words, the UDM is a database allowing to store the UEs' context and their states. The AMF is a worker reserved only to process procedures. So the AMF is stateless and any UE procedure can be handled by any AMF instance. When a request arrives to an AMF instance, the AMF loads the user context from the UDM, using the reference interfaces noted as Nx and depicted in Figure 1, processes the procedure and finally stores the new user context information and status in the UDM. This approach made the scalability of the AMF function easier and the architecture more flexible from a deployment point of view. Nevertheless, the decision of scaling, like how much instances to scale in/out or when to scale, is still missing.

Based on the ETSI NFV reference architecture, the scalability of a VNF is a decision taken by the NFV Orchestrator (NFVO), using the information provided by both the Virtual Infrastructure Manager (VIM) and the VNF Manager (VNFM). Therefore, for the sake of simplicity, we have only added a Service Orchestrator (SO) to the NGC architecture in this work. As mentioned earlier, our approach aims to balance the load between the AMF instances and to scale in/out the AMF functions as required. So the intelligence needed to apply this strategy is located in the SO.

IV. AMF Balancing and Scaling Proposal

In this section, we describe the used-methodology to derive fine grained predictions of the arriving traffic, which is later employed by the proposed load balancing model. Thus, we start by introducing the datasets, which were prepared for the learning phase. Then, we describe considered neural network models. Finally, we describe briefly the strategy used for load balancing.

A. Mobile Dataset and Data Classification

The dataset used in our work contains real mobile data collected by Telecom Italia Big Data Challenge [9] and hosted on Harvard Dataverse [10]. This dataset is rich of information. It contains two months of data collected on different Cells of the mobile core network of Mobile Italia in Milano, with a period of 10 minutes.

We assume in this paper that in each period of time 10% of the whole traffic (calls, sms and Internet traffic) is control traffic. The rest 90% of the traffic is considered as user data traffic. In order to use the data in our contribution (training the neural networks and testing them), the data followed some sorting and classification processes.

First, the data is sorted following the timestamps and not the cell ids. In other words, for each period of time (10 minutes), we regrouped together the data of all the cells in Milano city. Then we extracted our 10% of control traffic load with their timestamps.

Once we have the needed data, we propose to shift from a prediction problem into a classification one. This means that at each time stamp, the load data will belong to a load class. The load classes are distributed as following: the variation between the minimum control load and the maximum control load over the two months is split into 10 intervals. Thus the period between each interval of load is a class. Therefore, depending on the class, we are able to determine how much AMF instances we need to absorb, as we know already the capacity of our AMF (20 process per second in our case).

Then, each class is coded in order to increase the learning factor and the accuracy of our neural networks. The coding of the classes is simple and based on 'one-hot' coding. In other words, the class are coded using a word of 10 bits (as there are 10 classes), and the first class code will be the first bit equal '1' and the others equal '0'. The second class will have the second bit equals '1' and the other equal '0'... etc.

Finally to sum up, at the end of the data processing and classification, at each time stamp over two months, instead of having a number of connection, a class of load is defined where the number of connection fits in the interval of the class.

B. Neural Network Models

1) Deep Neural Network: As stated before, the scaling decision is based on prediction of the upcoming control traffic load using neural network models. In this contribution we have compared the prediction performance of two types of neural networks (DNN and RNN) using the mobile data classification listed above, in Section IV-A.

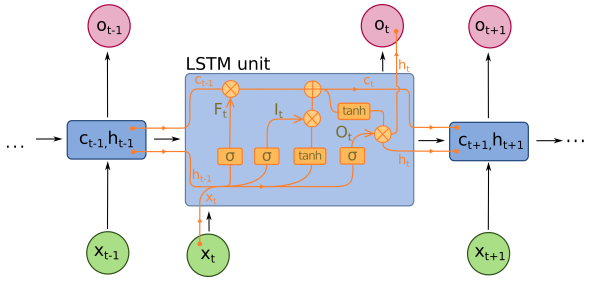

Fig. 2: Long Short Term Memory Cell Architecture

The DNN, or also called feed forward neural network, is composed of several neural layers. The DNN layers are classified as following: the input layer; the hidden layer(s); and the output layer. Each Layer is composed of nodes. Each node is like a neuron in our brain: Following a stimulus it may fire an action. The stimulus is actually the data *X* multiplied by a weight. Once we have stimuli, the stimuli are summed and passed to a function called 'The activation function'. The activation function detects the importance of the data and decides whether the data is propagated through the network in order to get the final outcome or vanishing it.

2) Recurrent Neural Network: The RNN is another type of neural networks. In our contribution, only LSTM cell [11] is used. The main difference between feed forward neural networks (i.e. DNN) and LSTM, is that the RNN keeps state memory of the last passed activation events in the network as temporal contextual information [12]. Figure 2 illustrates the structure of an LSTM cell. An LSTM cell is composed mainly of an input and three gates: the input gate, the forget gate and the output gate.

The input x_t is squeezed between -1 and 1 using the $tanh$ function and can be noted as following:

$$g_i^{(t)} = tanh(b_i^g + \sum_j U_{ij}^g x_j^{(t)} + \sum_j W_{ij}^g h_j^{(t-1)})$$

i and j respectively represent the cell and the neuron. b, U and W respectively denote the biases, input weights and recurrent weights into the LSTM cell. Then, the output of the input gate (a sigmoid gate), which is between 0 and 1 (I_t), is multiplied element-wise by $g_i^{(t)}$ in order to determine which inputs are switched on and off. This operation is denoted as $g \circ i$. Afterwards an inner state (C_{t-1}), providing an internal recurrence loop to learn the relationship between inputs separated by time, is ultimately added to the $g \circ i$.

Furthermore, the forget gate, a sigmoid function, is element-wise multiplied by C_{t-1} to determine which previous states should be remembered and which should be forgotten, allowing the LSTM cell to learn appropriate context. Finally, the output O_t of the sigmoid gating function is multiplied by the squeezed inner state s_t in order to get the LSTM cell output expressed as following:

$$h_t = tanh(s_t) \circ O_t$$

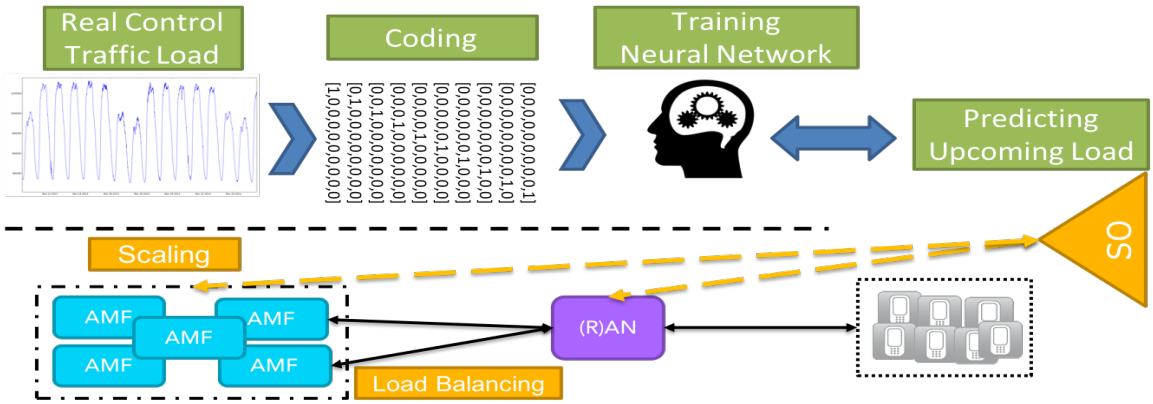

Fig. 3: Proposal Logic

For more details on RNN and LSTM structures and functions, interested readers may refer to [13].

C. Load Balancing

To equilibrate the load between the AMF instances, a control model developed and tested in a previous work [14] will be used. The used-controller is Linear Quadratic Regulator (LQR) [15]. The LQR aims to minimize the performance index in order to allow the system to converge to the goal with less controller action. So periodically, the LQR will calculate a feedback vector allowing to determine the relative capacity of each AMF, depending on its load. Once the relative capacities are calculated, they are pushed to the eNodeB in order to distribute the arrivals on the AMF instances depending on the load of each. For more details about the load balancing process please refer to [14].

V. Models Evaluation and Results Analysis

A. Simulator

In order to test the models, we developed a discrete event simulator in Python using Simpy [16]. The simulator simulates the behavior of UE arrivals that are buffered in the eNodeB queue. Then, the simulated eNodeB will dispatch the arrivals on the active AMF instances based on their relative capacity calculated periodically as mentioned above. The dispatched UEs will be hosted in the corresponding AMF queue and processed following a speed of 20 requests per second. This means that the latency chosen for a given AMF instance is $0,05$ milliseconds. Also, a given UE procedure will be rejected, due to the overload of the AMF instance, if the number of requests bypasses 50 requests in the AMF queue. In addition to the relative capacity calculation for each AMF, the SO checks each 10 minutes the neural network model in order to get the load class of the next period. Depending on the predicted class, the SO determines the needed number of AMF instances to be deployed for the next period. If the number of instances will increase, the SO will deploy the instances 60 seconds before the next period. The 60 seconds are considered as deployment latency. It is the needed-time to deploy an AMF instance in the data center and to be fully operational. The deployment latency (60 seconds) is calculated as an average following the tests conducted in [17]. Otherwise, for scaling in the AMF instances, the SO will order the removal of the over provisioned AMF instances 60 seconds after the next period in order to be sure that the load have decreased and avoid AMF overload. Finally, as the collected arrival data has a periodicity of 10 minutes, we distributed each period on 600 seconds in order to get more granularity and simulate a real traffic pattern. Therefore, the total number of arrivals of each period of 10 minutes is distributed following an inter arrival time, for each UE, calculated based on Poisson process law.

B. Scenarios

In order to test and compare our proposals, three different simulations were conducted. For the three simulations, we have considered the same arrival data (one week) with an AMF instance capacity of 20 requests per second. Also a deployment latency is considered over all the simulations equal to 60 seconds. The relative capacity refreshment of each AMF is achieved by the SO each 10 milliseconds.

1) Static Threshold Simulation: The first simulation consists on validating our simulator and showing the behavior of a dynamic scalability based on a static threshold, while using our control model to balance the load over the deployed AMF instances. In addition to the simulation parameters listed above, we consider a scale out threshold of 80% of load of the overall deployed AMF capacity. The scale in procedure is launched once less than 10% of the overall AMF instances capacity is used. The scalability decision is checked each 2 seconds.

2) Deep Neural Network Simulation: In this simulation we deployed a DNN composed of an input layer; 3 hidden layers and an output layer. The hidden layer 1 is composed of 500 neurons, the hidden layer 2 is composed of 1000 neurons and hidden layer 3 is composed of 1500 neurons. In order to train our DNN, we extract 60% of the dataset, shuffle them, then inject them into the DNN as batches of 500. From the 40% of data left, we selected one week of data and injected it in the simulator (the same data as the other simulations). So in this simulation, the SO checks each period of time (600 seconds) the predicted load class given by the DNN for the upcoming

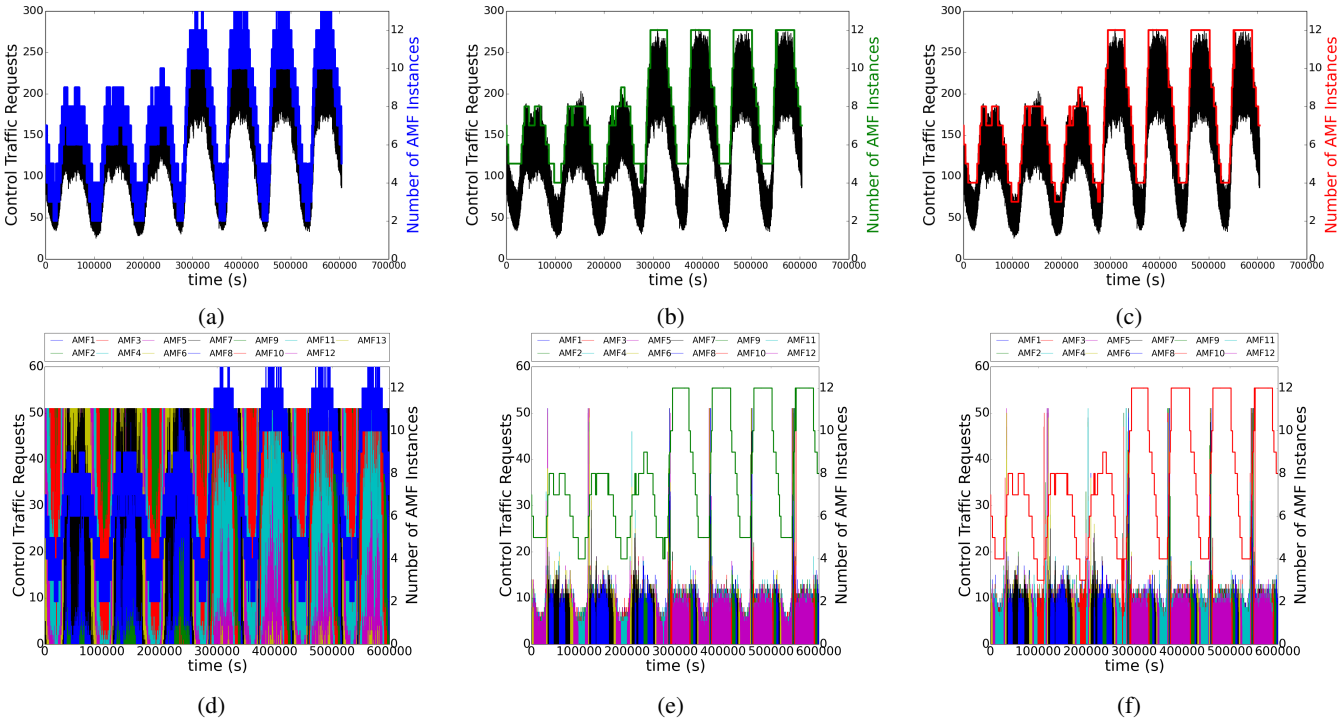

Fig. 4: Evolution of AMF instances number with control traffic Requests (a) threshold-based solution (b) DNN-based solution (c) RNN-based solution compared to the evolution of AMF instances load with AMF instances number (d) threshold-based solution (e) DNN-based solution (f) RNN-based solution

period of 600 seconds and decides whether to scale in/out the AMF instances.

3) Recurrent Neural Network Simulation: Finally, for this simulation we deployed the RNN composed of one LSTM cell. This cell is composed 256 neurons. As the DNN, 60% of the dataset were shuffled and injected as batches of 500 in the RNN. In order to keep the value of our data classification for the RNN and get the best prediction accuracy, each training data of a batch is composed of one RNN chunk of which has the size of the word of a coded class (10). Like in the DNN case, we selected one week of data from the 40% left in the dataset and inject it in the simulator(the same data as the other simulations).

C. Results Analysis

In Figure 4, we compare the number of deployed AMF instances in terms of the evolution of the load for (a) the threshold solution, (b) the DNN solution and (c) the RNN solution, and also the consequences on the AMF instances load in (d) threshold-based solution, (e) the DNN solution and (e) the RNN solution.

The threshold-based solution deploys up to 13 instances when the arrival rate is at the maximum rate as the RNN and DNN solutions deploy up to 12. On the other hand, when the load is at the minimum, the threshold-based solution deploys down to 2 instances, the DNN solution deploys down to 4 instances and finally the RNN deploys down to 3 instances. Although the threshold solution deploys more AMF instances than the RNN and the DNN solution, when the load is at its maximum, the AMF instances load is always reaching its maximum of 50 requests per seconds and thus the AMF instances are always overloaded. As for the DNN and RNN based solutions the AMF instances load is around 10 requests in the queue per seconds. This is explained by the fact that the threshold solution is reactive. Hence, it deploys the AMF instances after a load increase without foreseeing the time needed to provision new resources, which leads to AMF congestions and scaling oscillations.

On the other hand from Figure 5, we can see the consequences of overloading the AMF instances by evaluating the number of rejected requests: (a) Threshold-based solution, (b) DNN solution and (c) RNN solution, and the duration needed to process an attach request: (d) threshold solution, (e) DNN solution, (f) RNN solution. Comparing to the RNN and the DNN solutions, the number of blocked requests of the threshold-based solution is high. Also the attach duration of the UEs for the solution based on thresholds is always around 2.5 seconds as for the DNN and the RNN solutions is approximatively around 0.6 seconds. Hence, scaling in a proactive manner allow to avoid an increase of the latency while processing the control plane procedures.

Although the DNN is approximately similar to the RNN, it is worth mentioning that the RNN solution is slightly better that the DNN solution as it predicts more accurately (10% better than the DNN solution). This is shown in Figure 4, where we remark that when the load is at minimum, the

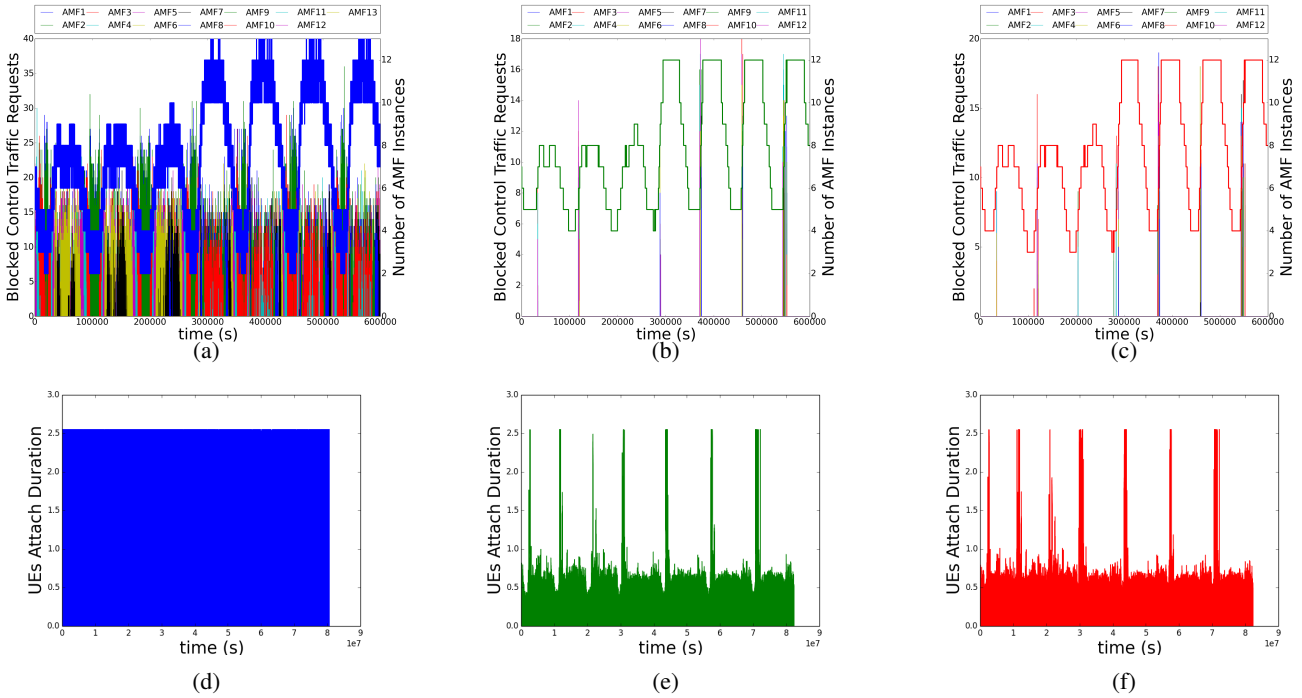

Fig. 5: AMF instances number vs Blocked traffic Requests (a) threshold-based solution (b) DNN-based solution (c) RNN-based solution compared to the evolution of the UEs attach duration (d) threshold-based solution (e) DNN-based solution (f) RNN-based solution

RNN based solution scales in down to two whereas the DNN solution scales in down to 3 AMF instances. Indeed, the RNN predicts more accurately and thus, helps in reducing the number of AMF instances to exactly what is needed in order to absorb the actual load.

VI. Conclusion

In this paper we proposed a novel solution to scale dynamically and pro-actively the VNF elements of the 5G new generation mobile core network. Our technique is based on neural networks where two different models are compared: the DNN and the RNN. Simulations, based on a real dataset of a Network Operator, showed that our solution scales in/out accurately the number of AMF instances depending on the load. Indeed, the proposed solution also bypasses the latency needed to provision new resources as it deploys in advance the needed number of AMF instances for the upcoming load; thus, avoiding overloading the core network and reducing the attach duration or the request process duration. We envision, for future work, to deploy our technique on a real platform in order to test its advantages on the field, and also we are willing to improve it in order to adapt when unexpected load pattern occurs.

References

[1] A. Nordrum, "Popular internet of things forecast of 50 billion devices by 2020 is outdated," IEEE Spectrum. [Online]. Available: https://goo.gl/5TS9s5

[2] I. Alawe, A. Ksentini, Y. Hadjadj-Aoul, P. Bertin, and A. Kerbellec, "On evaluating different trends for virtualized and sdn-ready mobile network," in *CloudNet*. IEEE, 2017, pp. 1–6.

[3] 3rd Generation Partnership Project (3GPP), "System architecture for the 5g system." [Online]. Available: https://tinyurl.com/yd9z3no9

[4] S. Dutta, T. Taleb, and A. Ksentini, "Qoe-aware elasticity support in cloud-native 5g systems," in *ICC*. IEEE, 2016, pp. 1–6.

[5] G. A. Carella, M. Pauls, L. Grebe, and T. Magedanz, "An extensible autoscaling engine (ae) for software-based network functions," in *NFV-SDN*. IEEE, 2016, pp. 219–225.

[6] A. B. Alvi, T. Masood, and U. Mehboob, "Load based automatic scaling in virtual ip based multimedia subsystem," in *CCNC*. IEEE, 2017, pp. 665–670.

[7] C. H. T. Arteaga, F. Rissoi, and O. M. C. Rendon, "An adaptive scaling mechanism for managing performance variations in network functions virtualization: A case study in an nfv-based epc," in *CNSM*. IEEE, 2017, pp. 1–7.

[8] A. Bilal, T. Tarik, A. Vajda, and B. Miloud, "Dynamic cloud resource scheduling in virtualized 5g mobile systems," in *GLOBECOM*. IEEE, 2016, pp. 1–6.

[9] T. Italia, "Telecom italia big data challenge." [Online]. Available: http://www.telecomitalia.com/tit/en/bigdatachallenge/contest.html

[10] ——, "A multi-source dataset of urban life in the city of milan and the province of trentino dataverse." [Online]. Available: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/EGZHFV

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

[12] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Interspeech*, 2014.

[13] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[14] I. Alawe, Y. Hadjadj-Aoul, A. Ksentini, P. Bertin, and D. Darche, "On the scalability of 5g core network: the amf case," in *IEEE CCNC*, 2018.

[15] K. Zhou, J. C. Doyle, K. Glover *et al.*, *Robust and optimal control*. Prentice hall New Jersey, 1996, vol. 40.

[16] Team-SimPy, "Discrete event simulation for python." [Online]. Available: https://simpy.readthedocs.io/en/latest/

[17] P. A. Frangoudis, L. Yala, and A. Ksentini, "Cdn-as-a-service provision over a telecom operators cloud," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 702–716, 2017.