# An Adaptive Scaling Mechanism for Managing Performance Variations in Network Functions Virtualization: A Case Study in an NFV-based EPC

Carlos Hernán Tobar Arteaga*, Fulvio Risso† and Oscar Mauricio Caicedo Rendon*
†Department of Automatics and Informatics, Politecnico di Torino, e-mail: fulvio.risso@polito.it
*Grupo de Ingeniería Telemática, Departamento de Telemática, Universidad del Cauca,
e-mail: {carlost, omcaicedo}@unicauca.edu.co

*Abstract*—**The scaling is a fundamental task that allows addressing performance variations in Network Functions Virtualization (NFV). In the literature, several approaches propose scaling mechanisms that differ in the utilized technique (*e.g.,* reactive, predictive and machine learning-based). The scaling in NFV must be accurate both at the time and the number of instances to be scaled, aiming at avoiding unnecessary procedures of provisioning and releasing of resources; however, achieving a high accuracy is a non-trivial task. In this paper, we propose for NFV an adaptive scaling mechanism based on Q-Learning and Gaussian Processes that are utilized by an agent to carry out an improvement strategy of a scaling policy, and therefore, to make better decisions for managing performance variations. We evaluate our mechanism by simulations, in a case study in a virtualized Evolved Packet Core, corroborating that it is more accurate than approaches based on static threshold rules and Q-Learning without a policy improvement strategy.**

## I. Introduction

The Network Functions Virtualization (NFV) enables to dynamically modify the capacity of Network Services (NSs) to face changes such as the number of users and performance variations [1], [2]. NSs are end-to-end functionalities created by composing Network Functions (NFs), virtualized or not [3]. The network performance allows assessing the quality of NSs; it is quantified by measurable parameters (*e.g.,* throughput and delay) [4], and its variation is associated with changes of underlying resources and the usage patterns of services and applications [5], [6]. For instance, in an Evolved Packet Core (EPC), a control entity sends and receives signaling messages for a proper operation [7]. The performance of such an entity can be measured in terms of the Mean Response Time (MRT) that can vary depending on the number of requests to establish sessions, update users location and perform handover [8].

The traditional solution to address performance variations is to oversize the capacity of NFs [9], which means that these are usually designed to support workload peaks. However, oversizing is inefficient at time slots of low utilization. NFV offers an alternative solution, the dynamic scaling of NFs that allows managing performance variations and improving the efficiency of using resources [10]. Scaling [11], the process of modifying the capacity of NFs, can be performed by increasing and reducing their resources (*i.e.,* scaling up/down) or creating and removing their instances (*i.e.,* scaling out/in). It is noteworthy that scaling can be initiated by administrators, as an outcome of the network performance assessment, or by the network itself by using adaptive mechanisms [12].

In the literature, several works have proposed NFV scaling mechanisms by using different techniques. For instance, [13] and [14] are based on threshold rules, a reactive technique, in which the scaling depends on the current traffic or performance, whose variations can lead to violations of a performance target and transitory scaling oscillations. This problem may be present also in solutions based on optimization, such as [15], which takes scaling decisions based only on instant traffic and it does not consider a prediction horizon to evaluate them. [16] uses time series forecasting that, based on historical data, enables to predict future resource usage; however, if there are changes in traffic patterns, an evolutionary strategy would be desirable, which would allow adapting the models to new conditions. [17] is based on the Q-Learning method in which, as other methods in Reinforcement Learning (RL), an agent interacts with an environment and learns by trial and error; this approach is adaptive but mistaken decisions may be taken until the agent learns an optimal scaling policy.

Considering the above limitations, we argue that the scaling in NFV should be adaptive and highly accurate to avoid violations of expected levels of Quality of Service (QoS) and transitory scaling oscillations. So, we consider that: (*i*) the use of RL for scaling is a good option since learning evolves whereas agents interact with their environment; and (*ii*) the scaling policy of Q-Learning can be iteratively improved before taking a final decision, and therefore the scaling could be more accurate. Here, we propose for NFV an adaptive scaling mechanism based on Q-Learning and Gaussian Processes (GPs), which are utilized by an agent to carry out an improvement strategy of a scaling policy, and therefore, to make better decisions for handling performance variations. By simulations, we evaluate our mechanism for managing variations of MRT in an NFV-based EPC, corroborating it is more accurate than approaches based on static threshold rules and Q-Learning without the policy improvement strategy.

The rest of this paper is organized as follows. Section II presents the related work. Section III introduces the proposed mechanism in an NFV-based EPC. Section IV evaluates and analyzes the behavior of our mechanism. And Section V provides some conclusions and implications for future work.

978-3-901882-98-2 © 2017 IFIP

## II. Related Work

In the NFV literature, several works have proposed scaling mechanisms that use different techniques. [13] and [14] use the technique of static threshold-based rules. Thresholds separate three performance regions: poor, good and oversized. If performance is in the good region no action is taken; if performance crosses from good to poor, scaling up/out starts. On the other hand, if performance crosses from good to oversized scaling down/in starts. In this technique, performance could exceed a target level at times that thresholds are crossed, which may incur in violations of expected quality. Also, if performance crosses consecutively thresholds, transitory scaling oscillations may happen. These transitory changes occur because scaling modifies the performance levels. Oscillations can lead to a significant problem because each scaling decision triggers procedures for provisioning or releasing resources.

[15] introduces optimization algorithms that minimize the power consumption and the cost of instantiating new NFs. These algorithms may also lead to violations of performance target levels and oscillations, since these algorithms consider only the current traffic and does not contemplate its prediction to evaluate the scaling decision to take. [16] proposes a mechanism that uses time series models for predicting CPU usage, and so, schedules required resources. This approach enables data efficiency, but it does not include an evolutionary strategy that would allow adapting the models to changes in the traffic patterns. [17] uses the Q-learning method that learns by trial and error in an environment. Learning implies to explore such an environment, and trying/erring means wrong scaling decisions; although these erroneous decisions occur just until the agent learns an optimal policy. We consider that this behavior must be avoided due to the implications on provisioning or releasing resources proper of each decision.

In summary, to face performance variations of NSs, the scaling decision-making should be adaptive and highly accurate to avoid performance violations, transitory oscillations, and wrong decisions. Our solution addresses this problem combining the Q-Learning method with GPs-based system models. Using the system models, an RL agent can iteratively improve its scaling policy, and therefore, to take more accurate scaling decisions.

## III. Scaling Mechanism

### A. A motivating scenario: NFV-based EPC

The current standard for 4th Generation (4G) mobile networks is the Long-Term Evolution (LTE), and its core is EPC [7]. The main entities of EPC are the Mobility Management Entity (MME), the Home Subscriber Server (HSS), the Serving Gateway (SGW) and the Packet Data Network Gateway (PGW). MME is the control entity responsible for signaling, mobility management of users, bearer management and QoS provisioning. HSS stores the administrative and user information utilized by MME. SGW and PGW compose the EPC data plane, which is in charge of routing and forwarding packets. For the proper EPC operation, SGW and PGW also interact with MME.

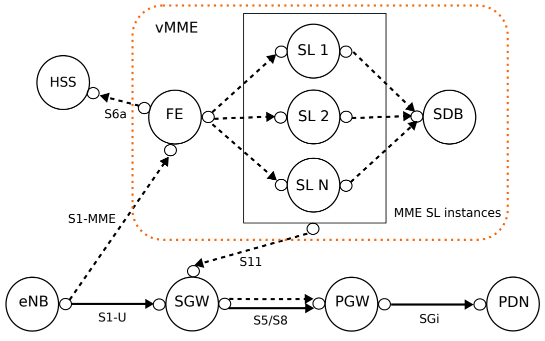

Fig. 1: Service Graph of an NFV-based EPC

From the Network Service Chaining (NSC) [18] point of view, EPC is composed of two network services: an NS for signaling (*i.e.*, control plane) and other NS for packet forwarding (*i.e.*, data plane). Figure 1 depicts these NSs by a Service Graph (SG) that follows the description proposed in [19]. The nodes: eNB and PDN are Service Access Points (SAPs) of SG. In particular, eNB is part of the access network and represents the base stations from where the mobile users are connected. In turn, PDN is any external data network like the Internet. Note that continuous arrows are service links that represent logical connectivity for the data plane, and dotted arrows are service links for the control plane. Each link is labeled with the protocol used. An SG is a directed graph depicting only one direction of the flow. However, a reverse chain is assumed since the communication is bidirectional.

In Figure 1, we consider a virtualized MME (vMME) [20] that can scale by increasing or decreasing the number of instances of its service logic, which enables to scale the control plane of EPC. For example, EPC may have more demand at evening than early morning depending on the number of service requests from users and, so, it is needed scaling vMME adaptively to improve the use of resources in the control plane. The vMME is formed by three components: Front-End (FE), MME Service Logic (SL) and State DataBase (SDB). FE acts as the communication interface with other entities of the network and balances the load among several MME SL instances that are in charge of processing control messages. SDB stores the user session state, hence enabling stateless SL.

### B. Adaptive Scaling Mechanism - Overview

Our mechanism aims at maintaining the MRT of the EPC control plane less than a particular threshold (*e.g.,* $1\ ms$ [20]). To satisfy such aim, the proposed mechanism can make scaling-out/in of vMME. In this sense, we consider that the infrastructure can instantiate up to $K$ instances (*e.g.,* $K = 4$) of vMME, and measure the offered workload and MRT.

Our mechanism is based on RL [21], a sub-field of machine learning, where an agent learns a decision-making process by interacting with an environment. From Markov Decision Processes [22], the agent and environment interact at discrete time steps. At each time step $t$, the agent receives some representation of the state of the environment, $S_t \in S$, where $S$ is the set of possible states. Based on $S_t$, the agent selects an action, $A_t \in A$, where $A$ is the set of available actions in the state $S_t$. One step later, the agent receives a numerical reward, $R_{t+1} \in \mathbb{R}$, and finds itself a new state, $S_{t+1}$.

Figure 2 depicts the RL process instantiated to our scaling problem; the environment is the NS of signaling in EPC, the states are pairs formed by the current number of instances and performance, and actions refer to the number of available instances. The steps, labeled as *1)* observe the state, *2)* take an action, and *3)* receive a reward, refer to a direct RL process called Policy Evaluation Process, which means that the agent interacts directly with the environment.

To calculate rewards, we consider the performance target (*e.g.,* MRT less than 1 $ms$) and the utilization factor $\rho = \lambda/(k \cdot \mu) < 1$ [23] of vMME, where $\lambda$ is the workload, $k$ is the number of instances and $\mu$ is the service rate of an instance. $\rho$ must be less than 1 to guarantee stability and allows defining expected ranges of utilization to be supported by vMME, which are $\frac{k-1}{k} \leq \rho < 1$. Given these ranges, the reward function is

$$R(k, \lambda) = \begin{cases} +1, & t_r < 1\ ms \ \wedge \ \frac{k-1}{k} \leq \rho < 1 \\ -1, & in\ other\ case \end{cases} \tag{1}$$

We use Q-Learning [24] as a method to perform RL. In this method, to each pair state-action is assigned an action value, which is the expected utility of carrying out an action $A_t$ when the agent is in the state $S_t$ and follows the most optimal policy. A policy is a rule for selecting actions. The value function is represented by $Q(S_t, A_t)$; and $Q$, implicitly defines the current policy $f$:

$$f_t(S_t) = a, \ such\ that\ Q_t\ (S_t, A_t) = \underset{A}{max} Q_t(S_t, A_t) \tag{2}$$

$f_t$ and $Q_t$ correspond to the policy and values of $Q$ at time $t$, respectively. That is, the current policy consists of choosing the action with maximal estimated value. The agent, through its experience, adjusts the values of $Q$ according to:

$$\begin{aligned} Q_{t+1}(S_t, A_t) \leftarrow (1-\alpha) Q_t(S_t, A_t) + \\ \alpha(R_{t+1} + \gamma \underset{A}{max} Q_t(S_{t+1}, A)) \end{aligned} \tag{3}$$

where $R_{t+1}$ denotes the reward received at step $t+1$, $\alpha$ is the *learning factor* (a small positive number) that allows the agent to retain what has been learned, and $0 \leq \gamma \leq 1$ is the discount factor that determines the importance of future rewards.

For learning, the agent needs to explore the environment and carry out the trial and error process; but, wrong actions lead to unnecessary procedures of provisioning and releasing of resources that must be avoided. It is here, where we use the system models. In this sense, for instance, in the EPC context, we consider the signaling workload as the input and MRT as the output of a dynamic system that is modeled by a regression function $h$. By observing these metrics, we train $h$ for predicting values of workload and use such predictions to run hypothetical iterations of the RL process, which allows getting the optimal policy before applying it to a particular NS. This is what we call Policy Improvement Process (see Figure 2).

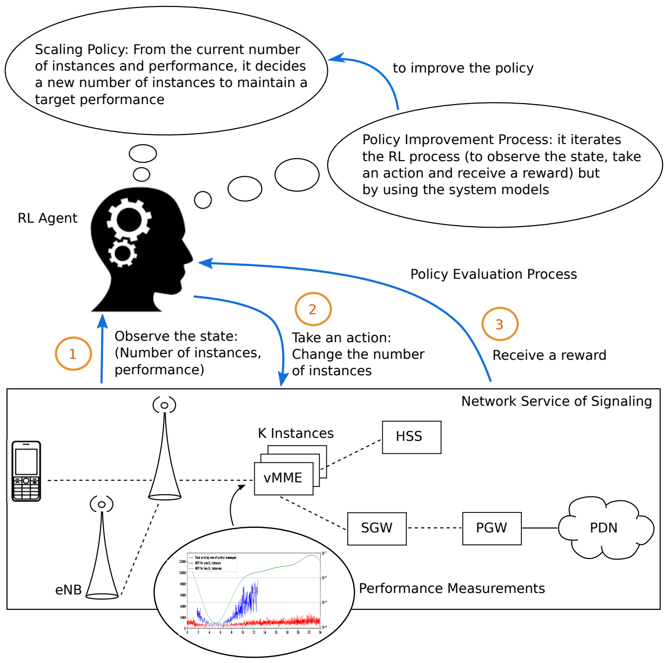
Scaling Policy: From the current number of instances and performance, it decides a new number of instances to maintain a target performance
to improve the policy
Policy Improvement Process: it iterates the RL process (to observe the state, take an action and receive a reward) but by using the system models
RL Agent
Policy Evaluation Process
1
Observe the state: (Number of instances, performance)
2
Take an action: Change the number of instances
3
Receive a reward
Network Service of Signaling
K Instances
HSS
vMME
SGW
PGW
PDN
eNB
Performance Measurements

Fig. 2: Overview of the Adaptive Scaling Mechanism

In summary, the agent runs two processes: the policy improvement and the policy evaluation; the first one allows the agent to foresee the results of its action, and the second one is the current execution of the policy. If there is no variation in the conditions of the NS (*e.g.,* number the users), which is reflected in the corresponding MRT, the agent probably just needs one iteration (to observe, take action and receive a reward) for improving its policy. But, if variations happen, the agent will need more iterations in the policy improvement to adapt to new conditions.

## *C. System Modeling*

We consider a network service as a dynamic system. For this system, we estimate a function $h$ given data $\mathcal{D}$: $\mathbf{x}_i \in \mathbb{R}^D$ (input) and $y_i = h(\mathbf{x}_i) + \varepsilon_i \in \mathbb{R}$ (output); the term $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is independent Gaussian measurement noise, which considers variations of $y_i$ in relation to the values of $h$. The estimation of $h$ refers to a regression problem that can be approached by parametric and non-parametric models. Parametric models impose a fixed structure on $h$ which limits its representational power. Non-parametric models allow determining the shape of the underlying function $h$ from the data and assumptions about its smoothness. Note that the term non-parametric does not imply models without parameters, but that the number of parameters is flexible and grows with the sample size.

We use regression based on GPs [25] that combines non-parametric models with Bayesian modeling and inference. A GP is fully specified by a mean function $m_h(\cdot)$ that describes how the average function is expected to look, and a covariance function which is also called a kernel

$$k_h(\mathbf{x},\mathbf{x}') = \mathbb{E}_h[(h(\mathbf{x}) - m_h(\mathbf{x}))(h(\mathbf{x}') - m_h(\mathbf{x}'))]$$
$$= cov_h[h(\mathbf{x}), h(\mathbf{x}')]$$

this function specifies the covariance between any two function values. Here, $\mathbb{E}_h$ is the expected value with respect to the function $h$.

We consider a Radial Basis Function (RBF) kernel [26] and a mean function $m_h = 0$. The RBF kernel, also known as the squared exponential kernel, has the form

$$k_h(\mathbf{x},\mathbf{x}') = e^{-\frac{\|\mathbf{x},\mathbf{x}'\|^2}{2\ell^2}} \tag{4}$$

where $||\cdot||$ represents the Euclidean norm and $\ell$ is the characteristic length-scale, which is a hyper-parameter that describes how smooth the function $h$ is; a small length-scale value means that values of $h$ can change quickly, while a large value characterizes $h$ that changes slowly.

Considering $h$ as a random function, Bayesian inference allows inferring a posterior distribution $p(h|\mathcal{D})$ over $h$ from the GP prior $p(h)$, the data $\mathcal{D}$ and assumptions on the smoothness of $h$. The posterior is used to predict $h(\mathbf{x}_*)$ values at arbitrary inputs $\mathbf{x}_* \in \mathbb{R}^D$. Briefly, Bayesian inference has three steps: ($i$) a prior on the unknown quantity has to be specified (in our case, $h$), ($ii$) data are observed; and ($iii$) a posterior distribution over $h$ is computed that refines the prior by incorporating evidence from the observations.

For instance, let's consider data $\mathcal{D}$ of workload and MRT, which are represented in vectors $\mathbf{x}, y \in \mathbb{R}$, respectively: $\mathbf{x} = [1000, 1200, 4600, 6400, 8200, 10000]^T$, $y = [0.0001, 0.0002, 0.00015, 0.0002, 0.00035, 0.002]^T$; $\mathbf{x}$ is measured in service requests per seconds and $y$ in seconds. Figure 3 (left) plots samples from the GP prior; the prior uncertainty about $h$ is constant (gray area) because there no observations. After having observed six function values (our data $\mathcal{D}$) represented by small circles in Figure 3 (right), samples from GP posterior depict that the uncertainty varies and depends on the location of the training inputs. In this example, we have chosen a length-scale $\ell$ of 3000, which allows to have the smoothness of $h$ in Figure 3.

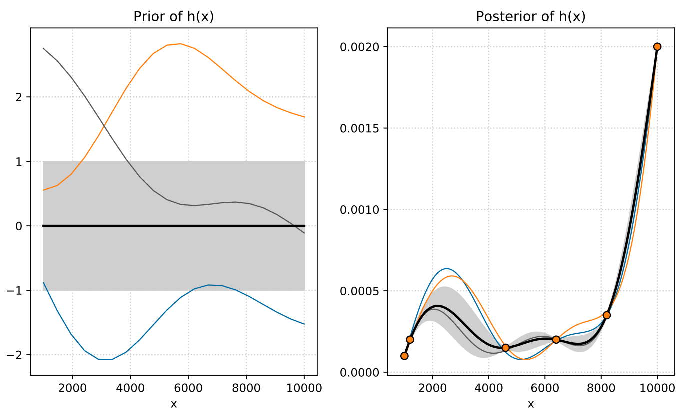

Fig. 3: Prior and Posterior of $h(x)$

## D. Scaling Processes

By using pseudo-codes, we detail the processes of policy improvement and policy evaluation.

**Algorithm 1:** Policy Improvement

```
Data: GP-based system models, Q-Learning parameters
      (e.g., $\gamma$ and $\alpha$), a reward function (e.g., see
      Equation 1) and a learning threshold (e.g., $\varepsilon_l$)
Result: An improved value function (i.e., Q) which is
        used by Algorithm 2
1  for each $t$ do
2      Initialize a variable $error$ to a value greater than $\varepsilon_l$;
         this variable allows finishing this iterative process;
3      while $error > \varepsilon_l$ do
4          Store $Q$ before the RL process in $previousQ$;
5          Get the scaling action $a$ using Equation (2);
6          Estimate $S_{t+1}$ using GPs-based system models;
7          Calculate the reward using Equation (1);
8          Update $Q$ using Equation (3);
9          Replace $S_t$ with $S_{t+1}$;
10         Store $Q$ after the RL process in $finalQ$;
11         Calculate the $error$ using the Mean Squared
             Error (Equation 5);
12     end
13 end
```

The Mean Squared Error (MSE) mentioned in the previous algorithm is given by

$$MSE = \frac{1}{N}\sum_{i=1}^{N} e_i^2 \tag{5}$$

where $e_i = previousQ - finalQ$ is the error between the value function before and after an iteration, and N is the number of elements of Q.

**Algorithm 2:** Policy Evaluation

```
Data: An improved $Q$ as result of Algorithm 1,
      Q-Learning parameters (e.g., $\gamma$ and $\alpha$) and a
      reward function (e.g., see Equation 1)
Result: A scaling-out/in action transferred to vMME
1  Since the agent receives its reward after that it takes
     action; this algorithm runs in two steps;
2  for each $t$ do
3      Measure and store the current state in $S_t$;
4      Get the scaling action $a$ using Equation (2);
5      Modify the number of instances of vMME according
         to the action $a$;
6  end
7  for each $t + 1$ do
8      Measure and store the current state in $S_{t+1}$;
9      Calculated the reward using Equation (1);
10     Update $Q$ using equation (3);
11 end
```

## IV. Evaluation and Analysis

By simulations, we evaluate the GPs-based system modeling and the behavior of our scaling mechanism. Also, we compare it with other ones: *(i)* based on static threshold rules, and *(ii)* based on Q-Learning without system models for improving the policy.

### A. System Modeling

We generate a synthetic workload and gather data of MRT from a simulated vMME. The synthetic workload is generated using the expressions given in [27], which consider that traffic in a mobile network exhibits a spatial-temporal pattern [28]. To measure MRT, we simulate the vMME of Figure 1 as a queuing model. Simulations followed a discrete event process implemented in Python [29]. As service rates, we use the defined in [20]: 120.000 packets per second for FE, 10.167 control procedures per second for MME SL and 100.000 transactions per second for SDB. Also, we consider a vMME composed by up to four SL instances; hence, four GPs-based models are needed for building the regression models of the vMME, one for each scaling configuration. We use scikit-learn [30] for creating GPs, tuning their hyper-parameters and performing predictions.

Figure 4 plots the signaling workload and some samples of MRT gathered from the simulated vMME (one and four instances for clarity), which are label as measured MRT. Also, in Figure 4 we plot MRT estimated from the models. Note that there is a good accuracy between measured and estimated MRT, which is confirmed by quantifying the MSE of the predictions (Equation 6): $8.6 \cdot 10^{-7}$ for one SL instance and $5.0 \cdot 10^{-7}$ for four SL instances.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} e_i^2 \tag{6}$$

where $e_i = measured\ MRT - estimated\ MRT$ is the prediction error, and N is the number of samples.

### B. Adaptive Scaling Behavior

We simulate an overall time of 24 hours, using as signaling workload the total arrival rate of control messages of Figure 4. We simulate the vMME of Figure 1 as a queuing model, and we use the GPs-based system models built previously. The simulation followed a discrete-events process in which by each time step the Policy Improvement (Algorithm 1) and Policy Evaluation (Algorithm 2) were executed. The time step was 10 minutes, which allows achieving a right level of granularity.

In our simulations, $\gamma$ is 0.8 because its value close to 1 allows both the agent to consider future rewards and the expected reward can converge. In turn, $\alpha$ is 0.1 because this small value enable the agent to retain the learning. A learning threshold $\varepsilon_l = 10^{-3}$ is a small enough difference between Q before and after an iteration of the policy improvement algorithm, which allows the loop to terminate.

Figure 5 presents the simulation results by plotting, in different scales, MRT and the number of SL instances vs time. These results reveal that MRT is smaller than its maximum allowed value (1 $ms$) all time, which corroborates the expected accuracy of our mechanism to determine the correct number of instances at the right time. The changes of the number of SL instances are performed at 0:40 AM (from four to three instances), 1:50 AM (from three to two instances), 3:10 AM (from two to one instance), 7:00 AM (from one to two instances), 9:00 AM (from two to three instances) and 6:07 PM (from three to four instances).

Let's review some internal details. Figure 6 illustrates the operation of our scaling mechanism by plotting the number of iterations per execution of the Policy Improvement (Algorithm 1). We can note that at the beginning, the policy improvement algorithm performs 160 iterations, which enables to the agent getting the initial policy. At times of change (0:40 AM, 1:50 AM, and so on), about 200 iterations are needed to improve policy because we are on the border between two states, but in the other times only one iteration is required. In summary, we can affirm that our mechanism adapts to changes in the environment and learns by using the GPs-based models. This strategy allows applying scaling actions to vMME only when the agent reaches an improved policy.

### C. Comparisons

Figure 7 presents the behavior of a mechanism that uses static threshold rules. For the shake of comparison, we use as rules those defined in our reward function (Equation 1). Note that this approach is accurate in the time to scale because of its reactive characteristic, however, when variations cross the thresholds several times, transitory oscillations happen, such as the occurred close to 2:00 AM. Also, non performance target compliances happen when a threshold is crossed. In short, our mechanism is better than mechanisms based on static threshold rules because it avoids transitory oscillations and violations of the performance target.

Figure 8 depicts the behavior of a mechanism that uses Q-Learning without system models for policy improvement. We maintain the same conditions that in our mechanism, this means, the parameters $\gamma = 0.8$ and $\alpha = 0.1$. Note that a decision to scale, from two to one SL instance, is taken around 4:00 AM, fifty minutes after our mechanism; this delay is caused by the small $\alpha$ value, which retains the previous learning of Q-Learning. A similar situation is observed between 7:00 AM and 9:00 AM. These delays cause that MRT exceeds its maximum value (1 $ms$). Other point to highlight is the transitory scaling oscillations that the agent makes when the vMME needs to be scaled, such as at 3:00 AM and 4:00 AM. These oscillations are because the agent tries wrong actions until it can achieve a good policy. In brief, our scaling mechanism is better than the based on Q-Learning without models for policy improvement. It is corroborated by the time when the scaling happens, the correct number of instances selected and the performance target compliance.

To sum up, our simulations confirm the expected accuracy of our approach. At each time step, the Q-Learning agent uses the system models for improving its scaling policy, and next, taking the best action based on that improved policy.

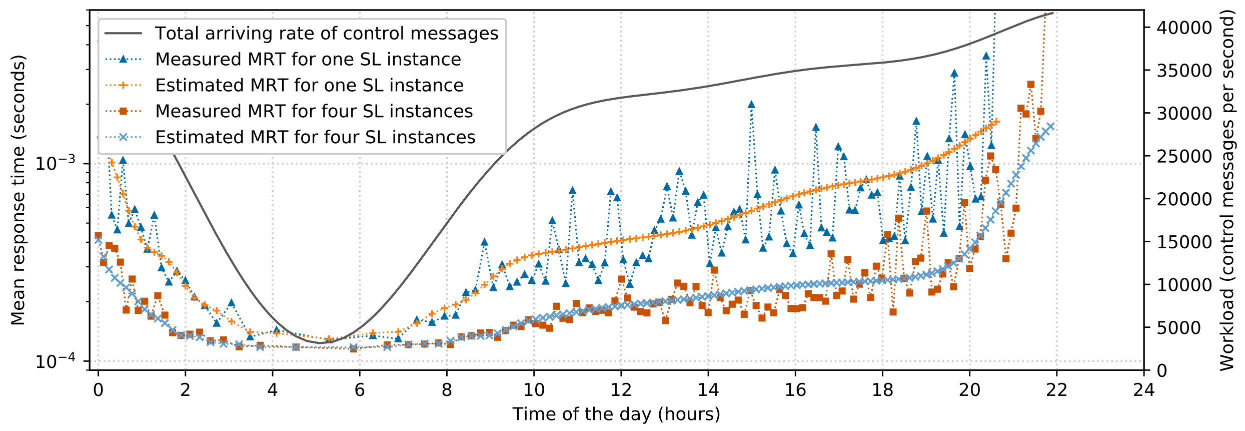

Fig. 4: Data samples of Workload, and Measured and Estimated Mean Response Time

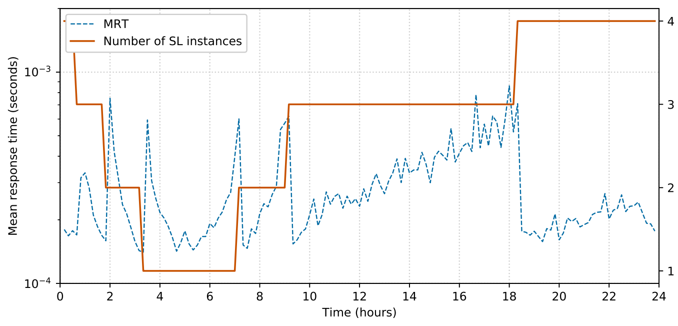

Fig. 5: Our Mechanism (Q-Learning + System Models)

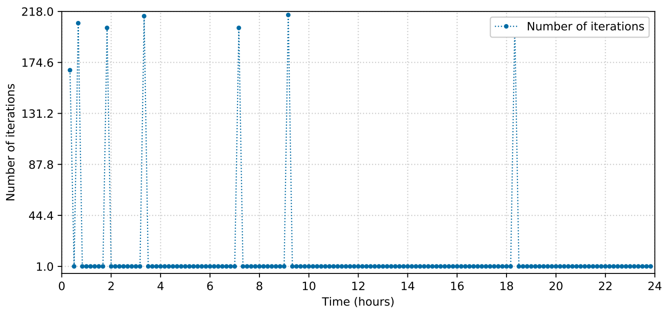

Fig. 6: Number of Iterations in the Policy Improvement

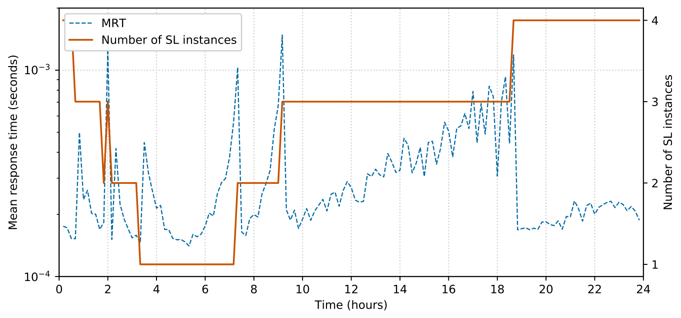

Fig. 7: Mechanism based on Static Threshold Rules

## V. Conclusions and Future Work

Network Functions Virtualization brings flexibility in deploying Network Services, allowing these services can be

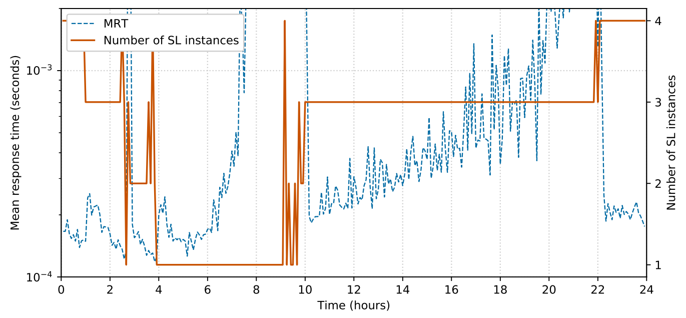

Fig. 8: Mechanism based on Direct Q-Learning

scaled depending on their workload demand or performance variations. In this paper, we have presented an adaptive mechanism that learns a scaling policy for managing network performance variations aiming at being accurate in the time for scaling and the correct number of instances to increase or decrease. The mechanism combines Q-Learning with Gaussian Processes-based system models that allow it to adapt to dynamic environments and improve its scaling policy before taking any action. We corroborate by simulations that our mechanism is more accurate than mechanisms based on static threshold rules, which are widely used, and Q-Learning without system models for improving its policy.

As future work, we will compare the GPs-based system modeling with parametric and non-parametric models, all applied to network performance modeling. Also, we intend to deploy our mechanism in an emulated and real test environment.

## Acknowledgment

The authors would like to thank the University of Cauca and the Administrative Department of Science, Technology and Innovation (COLCIENCIAS) - Colombia for supporting this research work through the PhD scholarship 527-2015 of the student Carlos Hernán Tobar Arteaga.

## References

[1] ETSI-GS-NFV. (2013) Network functions virtualisation (nfv); architectural framework. European Telecommunications Standards Institute. [Online]. Available: http://www.etsi.org

[2] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network functions virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, February 2015.

[3] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, November 2016.

[4] ITU-T-M800. (2008) Definitions of terms related to quality of service. Telecommunication Standardiation Sector of ITU. [Online]. Available: http://www.itu.int

[5] R. Shea, F. Wang, H. Wang, and J. Liu, "A deep investigation into network performance in virtual machine based cloud environments," in *IEEE Conference on Computer Communications (INFOCOM)*, Toronto, ON, Canada, 2014, pp. 1285–1293.

[6] F. Callegati, W. Cerroni, C. Contoli, and G. Santandrea, "Performance of network virtualization in cloud computing infrastructures: The openstack case," in *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, Luxembourg, Luxembourg, 2014, pp. 132–137.

[7] 3GPP. (2011) 3gpp ts 23.401: General packet radio service (gprs) enhancements for evolved universal terrestrial radio access network (e-utran) access. 3rd Generation Partnership Project. [Online]. Available: http://portal.3gpp.org

[8] A. Rajan *et al.*, "Understanding the bottlenecks in virtualizing cellular core network functions," in *IEEE International Workshop on Local and Metropolitan Area Networks (LANMAN)*, Beijing, China, 2015.

[9] P. Heidari and A. Kanso, "Qos assurance through low level analysis of resource utilization of the cloud applications," in *IEEE 9th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, USA, 2016, pp. 228–235.

[10] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Network function virtualization, state-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, Firstquarter 2016.

[11] ETSI-GS-NFV-MAN. (2014) Network functions virtualisation (nfv); management and orchestration. European Telecommunications Standards Institute. [Online]. Available: http://www.etsi.org

[12] ETSI-GS-AFI. (2013) Autonomic network engineering for the self-managing future internet (afi); generic autonomic network architecture. European Telecommunications Standards Institute. [Online]. Available: http://www.etsi.org

[13] G. Carella, M. Pauls, L. Grebe, and T. Magedanz, "An extensible autoscaling engine (ae) for software-based network functions," in *Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Palo Alto, CA, USA, 2016.

[14] S. Dutta, T. Taleb, and A. Ksentini, "Qoe-aware elasticity support in cloud-native 5g systems," in *IEEE International Conference on Communications (ICC)*, Kuala Lumpur, Malaysia, 2016.

[15] X. Wang, C. Wu, F. Le, A. Liu, Z. Li, and F. Lau, "Online vnf scaling in datacenters," in *IEEE 9th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, USA, 2016, pp. 140–147.

[16] A. Bilal, T. Tarik, A. Vajda, and B. Miloud, "Dynamic cloud resource scheduling in virtualized 5g mobile systems," in *IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, USA, 2016.

[17] P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang, "Efficient auto-scaling approach in the telco cloud using self-learning algorithm," in *IEEE Global Communications Conference (GLOBECOM)*, San Diego, CA, USA, 2015.

[18] W. John *et al.*, "Research directions in network service chaining," in *IEEE SDN for Future Networks and Services (SDN4FNS)*, Trento, Italy, 2013.

[19] J. Garay, J. Matias, J. Unzilla, and E. Jacob, "Service description in the nfv revolution:trends, challenges and a way forward," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 68–74, March 2016.

[20] J. Prados-Garzon *et al.*, "Modeling and dimensioning of a virtualized mme for 5g mobile networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 5, pp. 4383–4395, May 2017.

[21] R. Sutton and A. Barto, *Reinforcement learning: An introduction*, MIT, Ed., London, 2012.

[22] Mausam and A. Kolobov, *Planning with Markov Decision Processes*, Morgan and Claypool, Eds., 2012.

[23] M. Zukerman, *Introduction to Queueing Theory and Stochastic Teletraffic Models*, arXiv:1307.2968v14, Ed., 2016.

[24] C. Watkins, "Learning from delayed rewards," 1989.

[25] M. P. Deisenroth, *Efficient Reinforcement Learning using Gaussian Processes*, K. S. Publishing, Ed., 2010.

[26] M. Álvarez, L. Rosasco, and N. D. Lawrence, "Kernels for vector-valued functions: A review," *Foundations and Trends in Machine Learning*, vol. 4, no. 3, pp. 195–266, April 2012.

[27] S. Wang, X. Zhang, J. Zhang, J. Feng, W. Wang, and K. Xin, "An approach for spatial-temporal traffic modeling in mobile cellular networks," in *27th International Teletraffic Congress (ITC27)*, Ghent, Belgium, 2015, pp. 203–209.

[28] F. Xu, Y. Li, H. Wang, P. Zhang, and D. Jin, "Understanding mobile traffic patterns of large scale cellular towers in urban environment," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1147–1161, April 2017.

[29] Team-SimPy. Discrete event simulation for python. [Online]. Available: http://simpy.readthedocs.io

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.