
XIV Jornadas de Ingeniería Telemática.
JITEL 2019.
Universidad de Zaragoza.

Actas de las XIV Jornadas de Ingeniería Telemática (JITEL 2019), Zaragoza (España), 22-24 de octubre de 2019.

ISBN: X-XXXX-XX-XXX-X
DOI: XXXXXX

Lightweight Testbed for Machine Learning Evaluation in 5G Networks

Carlos Hernández-Chulde, Cristina Cervelló-Pastor
Department of Network Engineering,
Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain
{carlos.hernandez, cristina}@entel.upc.edu

***Abstract*—The adoption of Software Define Networking, Network Function Virtualization and Machine Learning will play a key role in the control and management of fifth-generation (5G) networks in order to meet the specific requirements of vertical industries and the stringent requirements of 5G. Machine learning could be applied in 5G networks to deal with issues such as traffic prediction, routing optimization and resource management. To evaluate the adoption of machine learning in 5G networks, an adequate testing environment is required. In this paper, we introduce a lightweight testbed, which utilizes the benefits of container lightweight virtualization technology to create machine learning network functions over the well-known Mininet network emulator. As a use case of this testbed, we present an experimental real-time bandwidth prediction using the Long Short Term Memory recurrent neural network.**

***Keywords*—5G, SDN, NFV, machine learning, containers**

I. INTRODUCTION

The fifth generation (5G) of communication networks will bring with new requirements, such as high data rates, high traffic densities, low latency and high reliability, and use cases such as the Internet of Things (IoT) and critical communication applications. These requirements and use cases impose new challenges, which demand efficient, intelligent and agile network management. Additionally, 5G will create an ecosystem that increases innovation opportunities for new applications in vertical industries such as manufacturing, healthcare, media and entertainment, financial services, public safety, the automotive industry, public transportation, energy utilities, food and agriculture, and city management. Each of these has a specific set of requirements in latency, throughput, availability, reliability, coverage, mobility, ans so on. 5G will provide a flexible network that caters to such varied requirements. Network flexibility implies a high degree of softwarization, virtualization and automation [1]. From the network perspective, Software Defined Networking (SDN) is considered to be the materialization of the softwarization concept, and Network Function Virtualization (NFV) of the virtualization paradigm [2]. One key component in enabling network flexibility is network slicing, as it allows us to create tailored logical networks on top of a common shared physical infrastructure in order to efficiently satisfy the specific needs of each vertical industry. A network slice involves a set of network functions and resources that are required to run these network functions. SDN and NFV can provide the programmability, flexibility and modularity that are necessary to create network slices [3].

Since SDN and NFV allow network functions to run in software instead of being tightly coupled with hardware, they provide flexibility and reconfigurability to the network. Thus, network functions can be modified, updated and placed at any location in the network. However, the dynamic behavior of network functions introduces complexities and makes the provisioning, management and control of network slices impractical in a manual way. In this dynamic environment, continuous monitoring and network analytics become compulsory to understand the network behavior. Similarly, providing the network with automation capabilities is essential for network operation and management. Network automation reduces operational costs, avoids human error and accelerates the service time to market.

Besides, the application of machine learning (ML) to network analytics provides the network with learning and decision-making capabilities. ML techniques can extract relevant information from the network data and then utilize this knowledge for autonomic network control and management, as well as service provisioning. Based on historical and real-time data, ML mechanisms can predict network behavior and adapt it to the new network conditions by allocating the required amount of network resources without overprovisioning. ML can also be used for energy-saving optimization. If the current demand is low, it may be possible to switch off some elements or migrate services to locations with lower energy costs in order to optimize energy consumption. ML may be

effectively applied in automatic network orchestration and network management, making self-organizing networks feasible. In other words, ML is a key enabler of automation and contributes to addressing the problem of deploying network intelligence. In this context, SDN and NFV combined with ML are key enablers of 5G networks [4].

In this respect, it is worth mentioning that standardization entities are working in this field. The 3rd Generation Partnership Project (3GPP) has introduced a Network Data Analytics Function (NWDAF) in the 5G System Architecture. NWDAF is defined as an operator-managed network analytics logical function that can provide slice-level network data analytics to a network function [5]. The European Telecommunications Standards Institute (ETSI) has created an Industry Specification Group (ISG) called Experiential Networked Intelligence (ENI). The ENI system is an innovative context-aware entity that enables intelligent service operation and management applying technologies, such as big data analysis and artificial intelligence mechanisms to adjust offered services based on changes in user needs, environmental conditions and business goals [6].

In this scenario, as ML has recently received much attention as a key enabler of the control and management of 5G networks, researchers need tools to design, test and evaluate it. Researchers face various difficulties when testing ML applications due to infrastructure limitations, the expense or difficulty in building physical testbeds, or the unavailability of emulation platforms. Thus, in this paper we present an emulation test platform that is able to emulate ML as network functions using Mininet and Docker containers to facilitate the development and testing of ML applications in 5G networks. Network functions are executed inside Docker containers that are interconnected through the underlying Mininet-based emulation environment.

The remainder of this paper is structured as follows. In Section II, theoretical background is reviewed. In Section III, we introduce the testbed architecture and detail its components. Section IV presents the results of experimentation results consisting of traffic prediction using the Long Short Term Memory (LSTM) recurrent neural network as a ML technique. Finally, in Section V, conclusions and plans for future work are presented.

II. BACKGROUND

A. Network Functions Virtualization (NFV)

NFV transforms the way in which operators design and manage networks by employing virtualization technology [7]. NFV decouples specialized network functions from hardware and implements them as Virtual Network Functions (VNFs). VNFs are implemented in software and deployed on commercial off-the-shelf (COTS) servers [8]. Multiple VNFs can be connected in order to create complex network services (NSs), which are managed by a management and orchestration system (MANO).

By separating network functions from hardware, NFV offers several advantages over traditional network architectures: (1) reduced equipment footprint and power consumption, as it is possible to collapse multiple network functions into a single physical server; (2) rapid service development and deployment, making network upgrade tasks easier; (3) longer hardware life cycles; and (4) reduced maintenance costs. These benefits mean that NFV enhances flexibility and scalability while reducing CApital and OPerational EXpenditure (CAPEX and OPEX) [7].

B. Containers

Since NFV involves implementing network functions in software, virtualization technologies such as virtual machines (VMs) and containers play an important role in VNFs' development. Prior to the deployment of VNFs in production environments, VNFs must be tested in confined, lightweight environments [9]. Researchers and developers use these environments in the development and prototyping of new NSs. In these environments workloads run as software instances over VMs or containers.

Containers and VMs provide application isolation and bundle applications with all of their dependencies in a self-contained unit that can run anywhere. Both share physical computing resources, allowing for efficient use in terms of energy consumption and cost. Although the goals of containers and VMs are similar, the approach to achieving them differs. While VMs provide hardware virtualizations, containers provide operating-system-level virtualization.

Containers are a more lightweight virtualization technology than VMs; unlike VMs, containers do not require a hypervisor, as VMs do. A VM also needs its own operating system, which means that each VM runs a full copy of an operating system, regardless of whether the operating system is the same on two or more VMs. This adds an overhead, as starting an operating system occupies time, memory and storage.

Containers run on the top of the host operating system, sharing the kernel. Applications running in containers share operating-system-level architecture that provides them with basic services. Containers require an underlying operating system that provides the basic services to all of the containerized applications. By sharing operating system resources, the need to replicate operating system code is significantly reduced, which means that a server can run multiple containerized applications with a single operating system installation. Therefore, containers are very lightweight in terms of size and starting time. In other words, this means that by using containers, we can run more application instances on a single server than we can with VMs.

Containers utilizes two kernel features, such as namespaces and control groups (cgroups), to create virtual environments on top of an operating system. Namespaces provide a layer of isolation by limiting what a container can view and access, such as processes trees, networking resources or file system. When a container runs, the kernel creates a separate namespace that the container will use. Thus, this container's access is limited to that namespace. In contrast, cgroups provide resource allocation. With

cgroups, the kernel create groups of processes for resource management purposes. Cgroups allow granular control over resources by limiting or prioritizing system resources such as CPU time, system memory, network bandwidth, or combinations of these. In this sense, cgroups ensure that the containers use the resources that they require [10].

C. GPU Usage in ML

As mentioned in Section I, ML will play a crucial role in the operation and management of 5G networks. Applying ML in network analytics enables the intelligent use of network-generated data. ML will provide the network with insights into traffic patterns, available resources, potential security threats and user behavior, allowing the network to proactively adapt or change its behavior based on previous knowledge of these issues.

In 5G, the number of devices connected to the network is expected to grow exponentially. Therefore, the amount of data collected for network control and management will also increase. Moreover, it is envisioned that 5G will involve a combination of different technologies such as heterogeneous networks, cloud computing or edge computing. In this complex ecosystem with large volumes and varied types of data, the application of ML in network analytics will require a significant computational power, which Graphical Processing Units (GPUs) can provide.

The execution of ML workloads can be accelerated by using GPUs. A GPU has more numerous and smaller cores than a CPU. As GPUs have many cores and each core performs rapid calculations simultaneously, they are highly suitable for parallel processing. Thus, the use of GPUs in ML is a cost-effective and high-performance option in comparison to traditional CPUs.

III. TESTBED ARCHITECTURE

The application of ML to provide the network with a certain degree of intelligence has attracted the attention of several standards bodies and industry forums. ML techniques enable the network to make autonomous decisions by processing large amounts of network data. As mentioned in Section I, 3GPP has included a dedicated function called NWDAF in 5G system architecture for the purposes of data collection and data analytics.

At this point, it is worth mentioning that 3GPP's 5G system architecture is service based. In a Service-based Architecture (SBA), the architecture elements are defined as network functions that offer their services via a common bus known as Service-Based Interface (SBI). Network functions that are allowed to make use of the provided services can directly communicate with each other as originators or consumers. The SBA model takes advantage of the latest virtualization and software technologies, such as containers, to offer modularity, extensibility, reusability and self-containment in network functions. NWDAF is one key function within SBA, facilitating access to network data analytics. Consumer network functions decide how the data analytics provided by NWDAF are used to improve the network performance. For example, the Policy Control Function (PCF) may use per slice data analytics in its policy decisions, or the Network Slice Selection Function (NSSF) may use the load-level analytic information for slice selection.

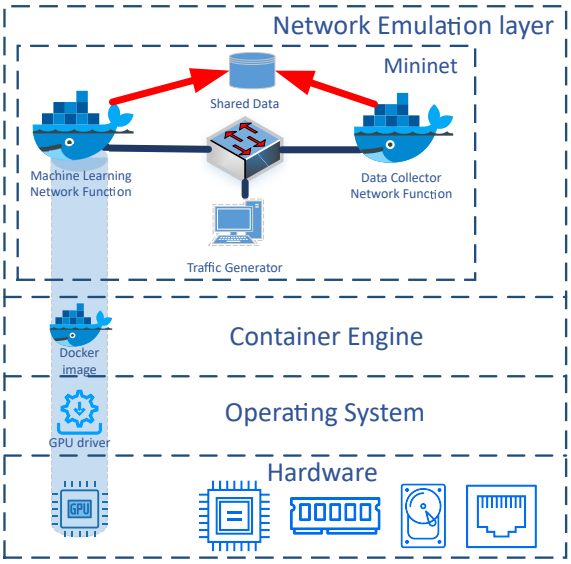


Fig. 1. Testbed architecture

In addition, the application of ML in network analytics requires both a module to monitor and collect data from the network and a module to apply ML techniques to extract knowledge from the data collected. The data collection module's role is to gather and store sufficient information from different sources to understand the current state of the network. It also performs data preprocessing to ensure that only useful data is stored. Relevant collected data may include network configuration, traffic data, control and management data, application and service-level data, and even external information, such as social networks [11]. The collected data are transformed into knowledge via ML in the second module. This module is the key component, as it is responsible for choosing the best ML algorithm that fits a certain problem or use case. In this module, based on real-time and historical network data, ML techniques can bring intelligence to the network by providing useful insights about its current and future states. The outcome of this module can be used by network controller or management systems to make decisions (either automatically or through human intervention) in order to optimize the use of network resources and enhance the provision of NSs.

In this context, it is necessary to have a platform ready for the development, prototyping and evaluation of network functions that provide network data analytics services such as NWDAF. The main objective of this work is to integrate the concepts and technologies described in Section II into a testbed, using the well-known Mininet network emulator and one of the most commonly-used container engines, Docker. The testbed architecture is ilustrated in Fig. 1.

In this testbed, data analytics network functions run as containerized applications within Docker containers.

Table I
TESTBED COMPONENTS

Component	Testbed Component
Network Emulator	Containernet
GPU Support for Containers	NVIDIA Container Runtime
Container Engine	Docker
Host Operating System	Ubuntu Bionic Beaver
GPU	NVIDIA GPU

There are two types of containers: one for data collection and one for ML application. The latter container type executes ML algorithms using GPUs. Since the used GPU is an NVIDIA GPU, the Docker containers use NVIDIA Container Runtime to access the GPU. NVIDIA Container Runtime simplifies the process of building and deploying containerized GPU-accelerated applications and guarantees the best performance on NVIDIA GPUs. Similarly, to provide interconnection between Docker containers on the top of Mininet, a fork of Mininet called Containernet is used [12]. Containernet extends the Mininet network emulator to allow the use of standard Docker containers as Mininet virtual hosts within the emulated network. In addition, Containernet allows the user to add or remove containers from the emulated network and to change resource limitation at runtime. Finally, any traffic generation tool, such as Iperf, can be used to generate traffic. The traffic generator host can be either a Mininet host or a Docker container.

This testbed, therefore, provides a framework for developing, testing and evaluating the application of ML in 5G networks in a simple, flexible and lightweight manner. Table I summarizes the components of this testbed.

IV. EXPERIMENTAL EVALUATION

We conducted an experiment to validate whether our testbed is lightweight and easy to use. The experiment consisted of predicting the traffic of a network via the use of ML; specifically, we used LSTM to do this. The experiment was carried out on a single physical machine which featured a CPU Intel(R) Core(TM) i9-9900K 3.60 GHz, 64 GB of RAM, running Ubuntu 18.04. The GPU used was an NVIDIA GeForce RTX 2080 with 2944 built-in cores and 8 GB of GDDR6 dedicated memory.

A. Traffic Prediction and LSTM

Network traffic prediction is an important issue in network operations and management, especially with regard to such diverse and complex networks as 5G networks. The aim of traffic prediction is to forecast the volume of future traffic by analyzing historical traffic information. Based on the results of traffic prediction, the network can make decisions in advance and adopt suitable preemptive actions to ensure its smooth operation, before a network overload occurs. These actions may include proactive routing policies or the provision of network resources.

Traffic prediction has been addressed via time series forecasting (TSF) [13]. Recent advances in deep learning have demonstrated that Recurrent Neural Networks (RNN) are powerful tools for TSF [14]. In our experimentation we used LSTM RNN for traffic prediction.

Time series traffic forecasting uses past traffic measurements to forecast future traffic patterns. For example, given a traffic measurement $x(t)$ at a time t, one can obtain a time series of $\{x(t), t = 1, 2, \ldots\}$. Traffic prediction consists of estimating the traffic at a future time $x(t+m)$ given n previous measurements, i. e.,

$$x(t+m) = f(\{x(n), n = 1, 2, \ldots, t\}) \quad (1)$$

LSTM is a special case and the most commonly used type of RNN. It is capable of learning long-term dependencies, which means that it can remember information that was previously learned. LSTM comprises multiple layers formed by one or more memory cells. A cell is responsible for memorizing values over time. Each cell is composed of three basic units: the input, output and forget gates that control information flow in an LSTM cell. The gates decide whether to forget, keep, update or output previously acquired information. LSTM is the most successful model for predicting long-term time series [15].

The input vector of our LSTM traffic prediction neural network corresponds to the recent traffic measurements, i.e., $x = [x(t), x(t-1), \ldots, x(t-n)]$, while the output vector is the predicted traffic in a future time $y = [x(t+1), x(t+2), \ldots, x(t+m)]$. Since LSTM networks retain past memory, traffic prediction for time interval $[t+1, t+m]$ is not only determined by the recent traffic measurements in $[t-n, t]$ but also indirectly by traffic measurements before $t-n$ through the memory cells.

B. Experimental Results

In our experiment, we deployed a simple topology over Mininet that consisted of a traffic generator host and the corresponding traffic sink, the data collector function and the ML function. All of these components were connected via a Mininet switch. Using Iperf, the traffic generator host generated traffic based on the dataset described later in this section. The data collector and ML network functions were Docker containers. One of the advantages of using Docker is that it offers us a repository of container images called the Docker hub. In this repository, we can find many containerized applications ready for use. As we intended to test ML algorithms, we used a container image that includes TensorFlow and CUDA. TensorFlow is an open-source platform for ML that provides a complete and flexible set of tools and libraries for ML development, whereas CUDA is a parallel computing platform which allows to harness the power of the NVIDIA GPUs, accelerating the ML workload execution.

To emulate the data collection function, we developed a Python script that periodically collected statistics from Mininet's switch interfaces and stored the collected data in a shared volume. The ML function accessed the collected data that were stored in the shared volume and used these data to train the LSTM RNN. Once the ML model had been trained, it was stored in the shared volume for later use in real-time traffic prediction. The training task is an

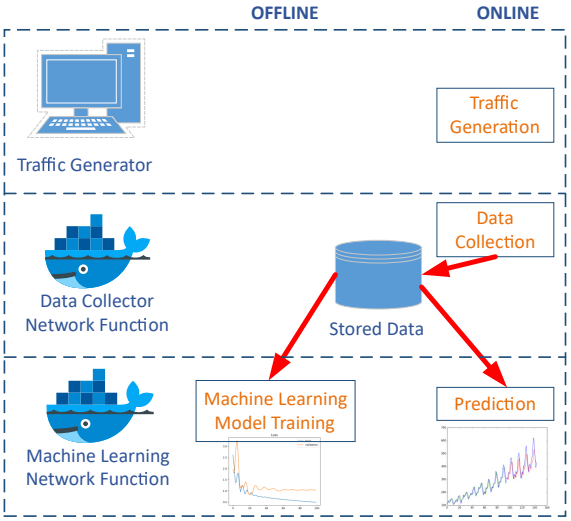
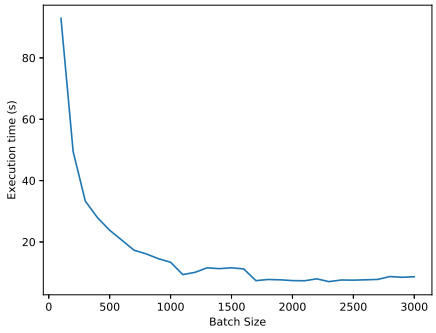
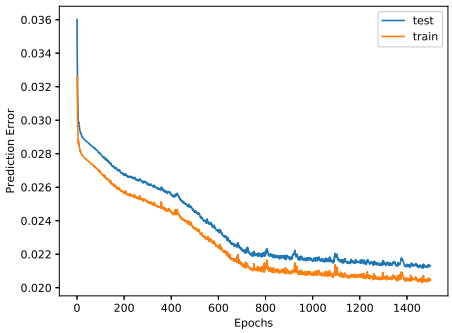


Fig. 2. Experiment details



(a) Batch size



(b) Epochs

Fig. 3. LSTM parameters

offline task that uses historical data for training LSTM model and can periodically retrain the model with the collected data, while the prediction task is an online task that makes prediction when a new traffic measurement is received. We also developed training and prediction tasks as Python scripts. It is worth mentioning that containerized network functions are executed on demand, which means that they run only for as long as it takes to execute the Python scripts execution, thus optimizing the use of computational resources. Fig. 2 presents the details of the experimental testbed.

The process of training our ML model is described below. We used a dataset from [16]. The dataset contained information about the traffic generated on a cellular network and provided hourly data on traffic statistics for each base station. The dataset consisted of (1) a base station identifier, (2) the date and time in UNIX format, (3) the number of users associated with the base station, (4) packets and (5) bytes transferred by the base station at the indicated time. In our case, we took the information of bytes and the date and time of two base stations to predict the traffic that each base station would use in the future.

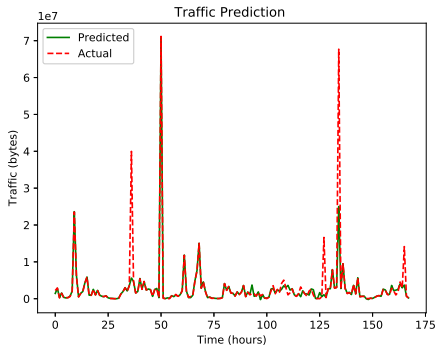
Specifically, we forecasted the traffic demand for each base station in the next hour, based on 24 past measurements ($n = 24$ (1 day)).

We generated training samples using a sliding window-based approach [14]. For example, to predict the traffic in the next hour ($m = 1$) based on the past 24 traffic measurements ($n = 24$), we used every consecutive 25 measurements as one training sample. The first 24 measurements became the input vector, and the 25^{th} measurement in the training sample was used as the output label.

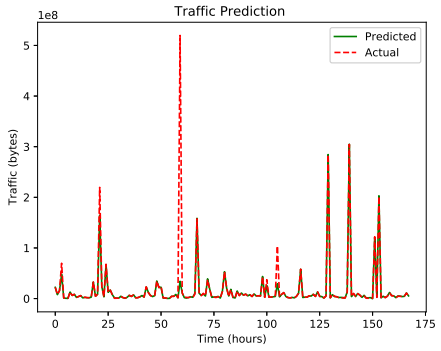
As the dataset was week-long, we reproduced the same data for the previous six months for training purposes. The dataset was divided in 80% training and 20% testing. Using the trained model, we predicted future traffic. Given that the LSTM architecture is characterized by the number of epochs and the batch size, we performed a set of experiments in order to identify the optimal values for these parameters to minimize the prediction error and execution time. The batch size is the number of training samples used in each iteration. We chose the value for the batch size that minimized the execution time. Fig. 3(a) shows that a batch size of above 1700 minimizes the execution time. In addition, the number of epochs determines the maximum number of passes over the training dataset. Different values for the number of epochs were tested in order to identify the optimal one that minimizes the prediction error. Thus, in the Fig. 3(b) it is evident that after 700 epochs, there is not a considerable improvement in prediction error, so this value was chosen in the LSTM.

We configured the LSTM network with one hidden layer, 100 neurons, an Adam optimizer with default values, 700 epochs and a batch size of 1700. Fig. 4 presents the traffic prediction results for the two base stations. The prediction values are very similar to the actual values, so this model was considered as a valid model for traffic prediction in this dataset.

In order to validate the computational overhead, we conducted offline training and online prediction, both in the CPU and GPU. In addition, to evaluate the overhead introduced by containers, we performed the same tasks on Docker containers and directly on the host. The training and prediction overhead in terms of processing time are presented in Table II. From the results in this table, it is evident that the training time is longer than the prediction time which is very short. However, this is not a problem



(a) Base station 1



(b) Base station 2

Fig. 4. Traffic prediction results

Table II
PROCESSING TIME FOR TRAINING AND PREDICTION

	Processing Time (s)			
	Host		Container	
	CPU	GPU	CPU	GPU
Training	14.350	9.195	14.390	9.312
Prediction	0.062	0.067	0.064	0.069

in traffic prediction, because training is an offline task and once the training is completed, the trained model can be used for real-time prediction.

As expected, the processing times for training and prediction on the containers and the host were similar; this is because containers can access the hardware directly through the operating system. Using containers does not lead to a virtualization overhead unlike in VMs with the hypervisor. Finally, using the GPU reduced training runtime, since GPUs allow parallel computing over a large number of cores, running thousands of threads at a time. GPU and CPU prediction times are quite similar, as prediction is a small workload and does not require a large number of threads. It is evident that the prediction time on the CPU is slightly lower than on the GPU, due to higher frequency of the CPU cores; the frequency in the CPU is 3600 MHz, whereas GPU's frequency is 1515 MHz.

V. CONCLUSIONS AND FUTURE WORK

This work presents the use of Docker containers and Mininet to build a lightweight testbed with the aim to evaluate the application of ML in 5G networks. In this testbed, the functions that perform network analytics using ML run as containerized NFVs. This paper also describes how containers can run ML algorithms on GPUs.

In our future work, we intend to integrate the testbed with an SDN controller and MANO system to test a comprehensive network ecosystem, in which the output of ML network functions will assist in the decision-making process to apply adequate policies and configuration parameters in the network. Likewise, we will use this testbed to assess the introduction of a distributed network analytics architecture for 5G networks applying distributed ML approaches.

REFERENCES

[1] A. Bosneag and M. X. Wang, "Intelligent network management mechanisms as a step towards 5G," in *2017 8th International Conference on the Network of the Future (NOF)*, Nov 2017, pp. 52–57.

[2] M. Condoluci and T. Mahmoodi, "Softwarization and virtualization in 5G mobile networks: Benefits, trends and challenges," *Computer Networks*, vol. 146, pp. 65–84, 2018.

[3] J. Ordonez-Lucena *et al.*, "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 80–87, May 2017.

[4] T. S. Buda *et al.*, "Can machine learning aid in delivering new use cases and scenarios in 5G?" in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, April 2016, pp. 1279–1284.

[5] 3GPP, "System Architecture for the 5G System," 3rd Generation Partnership Project (3GPP), Technical Specification (TS), Dec 2018.

[6] ETSI, "Experiential networked intelligence (ENI); ENI use cases," Experiential Networked Intelligence ETSI ISG, Group Report, Apr 2018.

[7] T. N. Tavares *et al.*, "NIEP: NFV Infrastructure Emulation Platform," in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, May 2018, pp. 173–180.

[8] ETSI, "Network Functions Virtualisation (NFV); Architectural Framework," Network Functions Virtualisation ETSI ISG, Group Specification, Dec 2014.

[9] S. van Rossem *et al.*, "Monitoring and debugging using an SDK for NFV-powered telecom applications," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016.

[10] RedHat. Introduction to Linux Containers. [Accessed: May 24, 2019]. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/overview_of_containers_in_red_hat_systems/introduction_to_linux_containers

[11] A. Mestres *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 47, no. 3, pp. 2–10, Sep. 2017.

[12] M. Peuster, H. Karl, and S. van Rossem, "Medicine: Rapid prototyping of production-ready network services in multi-pop environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.

[13] R. Boutaba *et al.*, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, Jun 2018.

[14] L. Mei *et al.*, "Realtime Mobile Bandwidth Prediction Using LSTM Neural Network," in *Passive and Active Measurement*, D. Choffnes and M. Barcellos, Eds. Cham: Springer International Publishing, 2019, pp. 34–47.

[15] A. Pelekanou *et al.*, "Provisioning of 5G services employing machine learning techniques," in *2018 International Conference on Optical Network Design and Modeling (ONDM)*, May 2018, pp. 200–205.

[16] [Online]. Available: https://github.com/caesar0301/city-cellular-traffic-map. [Accessed: Jun 1, 2019].