Deep Reinforcement Learning for Network Slicing with Heterogeneous Resource Requirements and Time Varying Traffic Dynamics

Jaehoon Koo
Northwestern University
Evanston, IL, USA
jaehoonkoo2018@u.northwestern.edu

Veena B. Mendiratta
Nokia Bell Labs
Naperville, IL, USA
veena.mendiratta@nokia-bell-labs.com

Muntasir Raihan Rahman
Nokia Bell Labs
Murray Hill, NJ, USA
muntasir.rahman@nokia-bell-labs.com

Anwar Walid
Nokia Bell Labs
Murray Hill, NJ, USA
anwar.walid@nokia-bell-labs.com

***Abstract*—Efficient network slicing is vital to deal with the highly variable and dynamic characteristics of traffic in 5G networks. Network slicing addresses a challenging dynamic network resource allocation problem where a single network infrastructure is divided into (virtual) multiple slices to meet the demands of different users with varying requirements, the main challenges being — the traffic arrival characteristics and the job resource requirements (e.g., compute, memory and bandwidth resources) for each slice can be highly dynamic. Traditional model-based optimization or queueing theoretic modeling becomes intractable with the high reliability, and stringent bandwidth and latency requirements imposed by 5G. We propose a deep reinforcement learning approach to address this dynamic coupled resource allocation problem. Model evaluation using synthetic and real workload data demonstrates that our deep reinforcement learning solution improves overall resource utilization, latency performance, and demands satisfied as compared to a baseline equal slicing strategy.**

***Index Terms*—network slicing, deep reinforcement learning**

I. Introduction

The challenges introduced by new technologies such as network function virtualization (NFV) [1] and software-defined networking (SDN) [2], and new network architectures such as 5G, are driving network transformations that radically change the way operators manage their networks and orchestrate network services. The network architectures come with a diverse range of capabilities and requirements, including massive capacity, ultra low latency, ultra high reliability, and support for massive machine to machine communications in the context of Industry 4.0 [3]. In order to consolidate multiple networks with varied requirements, the 5G architecture must exploit network virtualization and programmability. This introduces the problem of network slicing [4], where a single network infrastructure is divided into multiple sub-networks, and each slice can be operated by different parties. Each network slice represents an independent virtualized end-to-end network customized to meet the specific needs of an application, service, device, customer or operator [4].

A typical example used for 5G is as follows: deploy Internet of Things (IoT), Mobile Broadband (MBB), and vehicular communications applications on the same network, where IoT will typically have a large number of devices each with low throughput, MBB will have a smaller number of devices with high bandwidth content, and vehicular communications will have stringent latency requirements. The goal of network slicing is to enable partitioning of the physical network at an end-to-end level to allow optimum grouping of traffic, tenant isolation, and configuration of resources at a macro level.

Network slicing can be modeled as a dynamic resource allocation problem. Once the network is sliced into multiple sub-networks with SLA requirements on latency, bandwidth, reliability, etc., the underlying infrastructure operator needs to ensure that the SLAs for each slice are guaranteed under conditions of variability in the slice request arrivals and resource requirement distributions. While traditionally such network resource allocation problems have been solved using analytical queueing theoretic and optimization methods [5], [6], [7], given the complexity and scale of modern networks such methods are not feasible. Thus we resort to approximate black-box models using Reinforcement Learning (RL) [8]

Reinforcement learning is a computational approach for goal-directed learning and decision making, with the goal being to select actions to maximize future rewards [8]. The emphasis is on learning by an agent where each action influences the agent's future state, through direct interaction with its environment, without the need for exemplary supervision or complete models of the environment. RL can adapt the mapping from state to actions to maximize expected rewards in response to changing environmental conditions, for example, network environment, traffic dynamics, job size distribution, etc. Our hypothesis is that deep RL can be used to learn good

978-3-903176-24-9

network slicing strategies by learning over simulated and trace driven workload data, and such learned policies can be applied for real network slicing deployments.

We face three main challenges for efficient dynamic resource allocation for network slicing: unknown request arrival process, heterogeneous resource requirements for each slice, and finite resource capacities. While existing solutions deal with each challenge separately, we propose a unified solution using deep RL that can deal with these challenges simultaneously. In our formulation, each network slice has bandwidth and compute resource requirements, where the distributions of these requests are not known apriori. Our contributions are as follows: mathematical formulation of network slicing resource allocation as a Markov Decision Process (MDP); a policy-gradient method to solve this problem based on the REINFORCE [9] algorithm; and an experimental study with varying resource budgets using simulated and real datasets.

II. Background on RL and Deep RL

We give a brief introduction of RL next, specifically the use of neural networks for RL function approximation, and the policy gradient learning algorithm.

A. Basic Reinforcement Learning Model

In RL, an agent interacts with an environment. The agent has a set of actions to choose from, and the action can influence the next state of the environment. At each time step t, the RL agent observes a local copy of the environment's state s_t, and selects an action a_t. At the next time step $t+1$, the agent observes a reward r_t which represents the cost/reward for taking the last action. The agent also observes the next state s_{t+1}. We make the markovian assumption that the future state (s_{t+1}) only depend on the current state (s_t), and also that the dynamics are stationary. The agent's goal is to maximize the expected cumulative discounted return, $R_t = \mathbb{E}[\sum_t \gamma^t \cdot r_t]$, for $\gamma \in [0, 1)$, where the parameter γ is the discount factor.

B. Policy

The agent chooses actions based on a learned policy, which is a probability distribution over actions for any state, $\pi : \pi(s, a) \to [0, 1]$, where $\pi(s, a)$ denotes the probability of taking action a in state s, and $\sum_a \pi(s, a) = 1, \forall s$. In most practical scenarios, especially network environments, there are exponentially many possible (s,a) pairs (see Section III). Thus modern methods favor function approximators [8] that are scalable. A function approximator is parameterized by θ, and policies are denoted by $\pi_\theta(s, a)$. We use a deep neural network as the function approximator [8] in our models.

C. Policy Gradient Methods

We utilize the policy gradient methods class of policy learning algorithms that try to learn an optimal policy using gradient descent (or ascent) [8]. The objective is to maximize the expected cumulative reward; the gradient of this objective is, $\nabla_\theta \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]$, where Q^{π_θ} represents the expected cumulative discounted reward from selecting the action a in state s and then following policy π_θ. These methods estimate the gradient by sampling trajectories of policy executions and obtaining a reward estimate v_t for the trajectory, and subsequently update the policy parameters using gradient ascent with: $\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$, where α is the gradient ascent step size. This results in the REINFORCE algorithm [9], which we used for learning. The pseudo-code of the implemented training algorithm is shown in Section III.

III. Proposed Models

This section presents models for allocating bandwidth and Virtual Machines (VM) to network slices. We formulate the resource allocation problem, and describe it in RL settings for two service types: service upon arrival and batch service.

A. Service upon arrival

For service upon arrival, consider a network that receives resource requirements for bandwidth and VMs from K classes during the time horizon T. It is assumed that requests for bandwidth and VMs have the same arrival process for each class. However, the quantities of the resource requests have different and independent distributions. We assume that we have infinite buffers for both resources to hold received requests. The controller determines resource allocations for each class (slice) for the whole network at any arrival of each set of requests.

A set of bandwidth and VM requests for each class has a arrival process $\rho_i(t), i \in K, t \in T$. Each arrival has different amounts of arriving requests for bandwidth and VMs for each class. For bandwidth we have amounts of the requests, $x_i(t) \sim M_i(t)|\rho_i(t)$ and buffer levels $\beta_i(t), i \in K, t \in T$, where M_i is the distribution of the amounts for bandwidth requests. The buffer level can be computed by $\beta_i(t) = x_i(t) + \beta_i(t-1) - b_i(t-1)$. Similarly, for VMs, we have amounts of the requests, $y_i(t) \sim N_i(t)|\rho_i(t)$ and buffer levels $\delta_i(t), i \in K, t \in T$, where N_i is the distribution of the amounts for VM requests. Furthermore, we have resource allocations by a controller, $b_i(t)|x(t), \beta(t)$ and $v_i(t)|y(t), \delta(t), i \in K, t \in T$ for bandwidth and computing resources.

Our goal is to maximize the Quality of Service (QoS) with respect to bandwidth and VM requests and minimize resource costs, where QoS is defined as request processing delay; we solve the following optimization problem:

$$\text{minimize} \quad \mathbb{E}\big[\sum_{t=0}^{T} \gamma^t \mathcal{L}(\mathbf{b}(t), \mathbf{x}(t), \boldsymbol{\beta}(t), \mathbf{v}(t), \mathbf{y}(t), \boldsymbol{\delta}(t))\big] \quad (1)$$

where $\mathcal{L} = L_{\text{QoS}}(\boldsymbol{\beta}, \boldsymbol{\delta}) + w L_{\text{Res}}(\mathbf{b}, \mathbf{v})$, and L_{QoS} integrates delays in processing bandwidth and VM requests measuring buffer levels, L_{Res} is a cost for bandwidth and computing resources, and γ and w are discount and balance factors. This optimization problem is solved with RL algorithms. TABLE I shows the state, action and reward for the RL formulation.

Deep neural networks are introduced as a policy agent for determining the policy $\pi(s, a)$. A separate agent is assigned for each resource, namely CPU and bandwidth. The neural

Initialize θ_b, θ_v
for episode $= 1$ to all episodes **do**
  for $t = 1$ to $T - 1$ **do**
    if $\sum_{i=1}^{|K|} b_i^t \leq B^t$ **then**
      $b_i^t \leftarrow \frac{b_i^t}{\sum_{i=1}^{|K|} b_i^t} B^t$
    end if
    $\theta_b \leftarrow \theta_b - \alpha \nabla_{\theta_b} J_t$
    if $\sum_{i=1}^{|K|} v_i^t \leq C^t$ **then**
      $v_i^t \leftarrow \frac{v_i^t}{\sum_{i=1}^{|K|} v_i^t} C^t$
    end if
    $\theta_v \leftarrow \theta_v - \alpha \nabla_{\theta_v} J_t$
  end for
end for

Fig. 1: REINFORCE [10] Algorithm for Network Slicing

network agents feed the input feature vector: $< R, B, A >$, where R is the amount of received resource requests, B is the buffer level, and A is the last request arrival time, and computes the slice resource allocation as the output. The RL models are solved by applying a class of RL algorithms (known as REINFORCE [10]) using policy gradient methods that learn by performing gradient-descent on the policy parameters [8]. In the algorithm, J denotes the objective function, which is the expected cumulative discounted loss as shown in Eq. 1, and θ_b, θ_v are learning parameters of policy agents for each resource. Each neural agent consists of multiple hidden layers and one output layer. The leaky ReLU is adopted as an activation function at hidden and output layers with a small constant a.

$$h(x) = \begin{cases} x, & \text{if } x > 0 \\ ax, & \text{otherwise} \end{cases}$$

The leaky ReLU is selected as it produces positive allocations, and resolves a difficulty of ReLU when units are not active by allowing a small, positive gradient [11].

TABLE I: State, action and reward

	Service upon arrival	Batch service
State	Resource requests arrivals, and buffer levels since last arrival time	Resource requests arrivals, and buffer levels since last service time
Action	Resource allocation per slice	Resource allocation per slice
Reward	-(Request processing delay and resource use costs)	-(Request processing delay and resource use costs)

B. Batch service

Next we formulate the resource allocation problem for batch service, where requests are held in buffers, and the slice orchestrator periodically processes all the requests in the queue. The arrival process and assumptions are similar to the service upon arrival formulation; and the same optimization problem is solved. The MDP formulation is shown in Table I, however, a different neural network structure is proposed for a policy agent in the batch mode. For this mode, each deep neural network agent per class uses statistical measures (mean, max and standard deviation) of R (the amount of received resource requests), B (the buffer level), and A (the request arrival time) computed from the requests in a single batch. This allows the batch service to capture the dependencies among the requests in each batch, and prevents the allocation to simply sum up *per request* allocations. The hidden and output layers for the neural network agents in this model are similar to the service upon arrival model.

We compare our proposed algorithms to an ES policy which fairly divides the resources among each slice. Such a policy only makes sense when there is a finite budget for resources. Thus in our RL algorithms, we also impose budget constraints for each resource type to make a fair comparison, however our proposed algorithms are generic and can also work when there is no bound on resource capacity. We introduce budget constraints to Equation (1) as: $\sum_{i=1}^{|K|} b_i \leq B$, and $\sum_{i=1}^{|K|} v_i \leq C$, where B and C are the budgets for bandwidth and compute resources respectively.

To implement budget constraints, we project obtained allocations from neural agents proportionally when the sum of the allocations exceeds the budgets, that is, $b_i \leftarrow \frac{b_i}{\sum_{i=1}^{|K|} b_i} B$ and $v_i \leftarrow \frac{v_i}{\sum_{i=1}^{|K|} v_i} C$.

IV. Performance Evaluation

We present the scenarios for experimentation and the results obtained. Though we experimented with simulated and real data, only the results for real data are shown due to space constraints. All model implementations are done in Python TensorFlow using the GeForce GTX 1080 and Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz.

A. Data for Models

Two workload traces for CPU and bandwidth requests were used for algorithm evaluation. For job arrival times and job sizes (input bytes) we utilize the SWIM [12] workload suite for Facebook map-reduce cluster traces using 3 different traces for each slice. Each workload is for 1 hour of production cluster usage. For bandwidth data we use a 4G LTE trace from [13] by sampling the bandwidth at specific instances. The CPU and bandwidth traces are combined as follows: for each arrival time in the Facebook trace, we take the maximum bandwidth value in the interval $[current\ job\ arrival\ time\ -\ previous\ job\ arrival\ time]$.

B. Scenarios

For analysis, we create scenarios with 4 levels of the resource budget: smaller, small, large, and larger. Each arriving request requires two resources: bandwidth and compute (number of VMs). The budget size is determined using the mean and standard deviations of the resource request distributions. For service upon arrival, $B = \sum_{i=1}^{|K|} \mu_i + c\,\sigma_i$, and $C = \sum_{i=1}^{|K|} \nu_i + c\,\eta_i$, where (μ_i, σ_i) and (ν_i, η_i) for class i are the mean and standard deviation respectively of request distributions for the bandwidth and VMs. c is set as 0, 1, 2, and 3 for scenarios of smaller, small, large, and larger resource budgets respectively. Batch service requires a larger budget

since arriving requests wait in the buffer till the next batch service time. Thus, we multiply the service time interval and the request arrival rate to the budget of the service upon arrival such that $B_{\text{batch service}} = \sum_{i=1}^{|K|} \tau_i \lambda_i (\mu_i + c\sigma_i)$ where, for class i, τ_i is the service time and λ_i is the arrival rate.

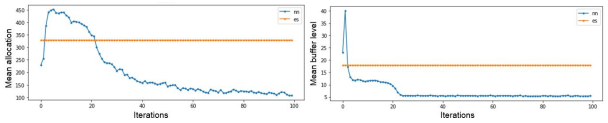
C. Real Data

The models are validated by conducting trace driven experiments using the data described in Section IV-A. For bandwidth requests, it is assumed that the maximum size of the request amounts since the last arrival should be serviced. Given the limited number of traces we assume that the requests are recurrent. To prevent divergence during training, we appropriately scaled down the original values for compute and bandwidth in the traces. The traces are split into training (90%) and test sets (10%). For the hyper-parameters, different settings were tested: learning rates from 0.1 to 0.001, the number of layers from 1 to 3, and 500 and 1000 units per layer. Based on the results the following settings were chosen. For service upon arrival, we generate 1000 and 100 episodes for training and test respectively. For batch service, we generate 5000 and 100 episodes for training and test respectively. The service time interval is set at 10 for batch service. We put the same weights on delays and resource use costs, $w = 1$. In both cases, each neural agent has 3 hidden layers with 1,000 units. The Adam optimizer [14], an algorithm for first-order gradient-based optimization of stochastic objective functions, is used for the gradient optimization using a learning rate of 0.001.

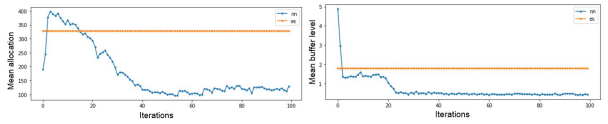
Table II shows the results of the proposed models (**NN**) compared with the ES strategy (**ES**), in terms of the expected rewards for each class and resource. The winners are shown in **bold**. The results show that our models perform better in almost all the scenarios as compared to the ES strategy. Though the NN models may have less rewards in individual cases, in total the NN models earn larger rewards. For example, in the small budget scenario for service upon arrival our model has a greater loss for BW allocations in classes 1 and 3 as compared to the ES strategy, however, it achieves a smaller total loss due to the savings in class 2. This can be explained by looking at the training phase shown in Figure 2, in that our models learn the request amount distributions of all classes and then allocate resources accordingly. During training, our models allocate resources differently to each class, and buffer levels also decrease. Based on experimental results we can conclude that our models can learn efficient resource allocation policies for the network slicing dynamic resource allocation problem and outperform the baseline strategy.

TABLE II: Real data: Loss (-reward) and winners

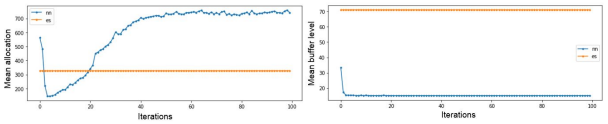
			Service upon arrival			
		Class	C1	C2	C3	Total
Small budget	BW	NN	4.585E+02	**1.878E+03**	6.056E+02	**2.942E+03**
		ES	**2.387E+01**	1.318E+05	**1.793E+01**	1.319E+05
	VM	NN	**9.702E+01**	**9.266E+01**	**5.609E+02**	**7.506E+02**
		ES	1.718E+02	1.719E+02	9.897E+02	1.333E+03
Larger budget	BW	NN	**7.521E+00**	**1.321E+01**	**7.910E+00**	**2.864E+01**
		ES	9.548E+00	4.866E+01	9.548E+00	6.775E+01
	VM	NN	**8.966E+01**	**1.095E+02**	1.271E+03	**1.470E+03**
		ES	4.757E+02	4.757E+02	**7.341E+02**	1.685E+03
			Batch service			
		Class	C1	C2	C3	Total
Small budget	BW	NN	1.283E+04	**1.594E+04**	9.770E+03	**3.854E+04**
		ES	**1.108E+04**	2.081E+04	**6.741E+03**	3.862E+04
	VM	NN	**8.109E+02**	**6.816E+02**	2.009E+03	**3.501E+03**
		ES	1.163E+03	1.163E+03	**1.342E+03**	3.669E+03
Larger budget	BW	NN	3.229E+03	**5.697E+03**	2.654E+03	**1.158E+04**
		ES	**2.109E+03**	1.169E+04	**7.079E+01**	1.387E+04
	VM	NN	**2.670E+03**	**2.145E+03**	4.919E+03	**9.734E+03**
		ES	3.245E+03	3.245E+03	**3.252E+03**	9.741E+03



(a) Resource allocation: Class 1 (b) Buffer level: Class 1



(c) Resource allocation: Class 2 (d) Buffer level: Class 2



(e) Resource allocation: Class 3 (f) Buffer level: Class 3

Fig. 2: Real data: Mean resource allocations, and buffer levels of three classes (large budget, service upon arrival, VM)

V. Conclusions

There is considerable literature on network resource allocation problems including network slicing using RL and deep RL approaches; these are categorized in Table III. The work in [15] is closest to our work — it formulates the network slicing problem with deep RL for two resource management scenarios: radio resource slicing for a base station, and priority-based core network slicing. However, our work is different from [15] in several critical aspects — we address the issue of allocating multiple resources simultaneously, solve constrained problems by introducing buffers, consider both service upon arrival and batch service, and validate model performance with simulated and real data.

TABLE III: Current approaches for resource allocation

	Resource Allocation	Network Slicing
RL	Bandwidth Allocation [16], [17], [18], [19], [20] Compute Allocation [21], [22]	Q-learning [23] Genetic optimization [24]
Deep RL	Cognitive radio networks [25] Cloud radio access networks [26] Vehicular ad hoc networks [27]	Deep RL for network slicing [15]

To summarize, we have proposed a new deep RL framework for network slicing with heterogeneous resource requirements and finite capacity which can deal with dynamic traffic demands from network users. Simulation and real trace experiments show that our system outperforms the baseline.

REFERENCES

[1] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De 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, 2016.

[2] N. McKeown, "Software-defined networking," *INFOCOM keynote talk*, vol. 17, no. 2, pp. 30–32, 2009.

[3] Wikipedia contributors, "Industry 4.0 — Wikipedia, the free encyclopedia," 2019. [Online; accessed 22-April-2019].

[4] "An introduction to network slicing," *GSM Association*, 2017.

[5] M. J. Neely, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan and Claypool Publishers, 2010.

[6] S. Shakkottai and R. Srikant, "Network optimization and control," *Found. Trends Netw.*, vol. 2, pp. 271–379, Jan. 2007.

[7] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control and Stochastic Networks Perspective*. New York, NY, USA: Cambridge University Press, 2014.

[8] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–, Oct. 2017.

[10] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, pp. 229–256, May 1992.

[11] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," *International Conference on Machine Learning*, 2013.

[12] "Statistical workload injector for mapreduce (swim)." https://github.com/SWIMProjectUCB/SWIM/wiki. Accessed: 2018-01-14.

[13] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. D. Turck, "Http/2-based adaptive streaming of hevc video over 4g/lte networks," *IEEE Communications Letters*, vol. 20, no. 11, pp. 2177–2180, 2016.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[15] Z. Zhao, R. Li, Q. Sun, Chi-Lin, C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep reinforcement learning for network slicing," *ArXiv*, 2018.

[16] A. Elwalid, D. Mitra, and R. H. Wentworth, "A new approach for allocating buffers and bandwidth to heterogeneous, regulated traffic in an atm node," *IEEE Journal on selected Areas in Communications*, 1995.

[17] D. Hetzer, "Adaptable bandwidth planning using reinforcement learning," *Journal of Systemics, Cybernetics and Informatics*, 2006.

[18] H. Tong and T. X. Brown, "Adaptive call admission control under quality of service constraints: a reinforcement learning solution," *IEEE Journal on Selected Areas in Communications*, 2000.

[19] T. C.-K. Hui and C.-K. Tham, "Adaptive provisioning of differentiated services networks based on reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and Reviews*, 2003.

[20] J. C. Ernst Nordström, "A reinforcement learning scheme for adaptive link allocation in atm networks," *Rural Growth Linkages in the Eastern Cape Province of South Africa*, 1995.

[21] P. Jamshidi, A. Shariflooz, C. Pahly, A. Metzgerz, and G. Estrada, "Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution," *International Conference on Cloud and Autonomic Computing*, 2015.

[22] J. V. B. Benifa and D. Dejey, "Rlpas: Reinforcement learning-based proactive auto-scaler for resource provisioning in cloud environment," *Mobile Networks and Applications*, 2018.

[23] D. Bega, M. Gramaglia, A. Banchs, V. Sciancaleporey, K. Samdanisz, and X. Costa-Perez, "Optimising 5g infrastructure markets: The business of network slicing," *IEEE Conference on Computer Communications*, 2017.

[24] B. Han, J. Lianghai, and H. D. Schotten, "Slice as an evolutionary service: Genetic optimization for inter-slice resource management in 5g networks," *IEEE Access*, 2018.

[25] Y. He, F. R. Yu, N. Zhao, V. C. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Communications Magazine*, 2017.

[26] Z. Xu, Y. Wang, J. Tang, J. Wang, and M. C. Gursoy, "A deep reinforcement learning based framework for power-efficient resource allocation in cloud rans," *IEEE ICC 2017 Next Generation Networking and Internet Symposium*, 2017.

[27] Y. He, F. R. Yu, N. Zhao, H. Yin, and A. Boukerche, "Deep reinforcement learning (drl)-based resource management in software-defined and virtualized vehicular ad hoc networks," *Association for Computing Machinery (ACM)*, 2017.