Deep Reinforcement Learning for RAN Optimization and Control

Yu Chen, Jie Chen, Ganesh Krishnamurthi, Huijing Yang, Huahui Wang, Wenjie Zhao

Abstract—Due to the high variability of the traffic in the radio access network (RAN), fixed network configurations are not flexible to achieve the optimal performance. Our vendors provide several settings of the eNodeB to optimize the RAN performance, such as media access control scheduler, loading balance, etc. But the detailed mechanisms of the eNodeB configurations are usually very complicated and not disclosed, not to mention the large KPIs space needed to be considered. These make constructing simulator, offline tuning, or rule-based solutions difficult. We aim to build an intelligent controller without strong assumption or domain knowledge about the RAN and can run for 24/7 without supervision. To achieve this goal, we first build a closed-loop control testbed RAN in a lab environment with one eNodeB provided by one of the largest wireless vendors and four smartphones. Next, we build a double Q network agent that is trained with the live feedbacks of the key performance indicators from the RAN. Our work proved the effectiveness of applying deep reinforcement learning to improve network performance in a real RAN network environment.

Index Terms—self-organizing network, radio access networks, deep reinforcement learning, testbed

I. INTRODUCTION

The self-organizing network (SON) is proposed to automate the planning, optimizing, and healing processes of the radio access network (RAN) [1]. As the mobile network and the traffic demands grow dramatically over the past years, it becomes critical to proactively self-optimize the configurations of the network, to minimize human intervention in development processes, and to improve the performance. Due to the high variability of the network status and dynamics of the connected user equipments (UE), network configurations using fixed rules are not flexible to achieve the optimal performance.

There are several key challenges. First, designing rule-based algorithms requires strong domain knowledge, and such a design can not be flexibly transferred to a new network with even slightly different settings. The development cycle of each rule-based algorithm could last from months up to years and involves a significant amount of manual work. The procedure highly depends on individuals, which makes the model less reliable and reproducible. Besides, it is hard for humans to capture the transient and subtle changes in the RAN, so some important details can be lost. Second, it is challenging to find the optimal RAN model of high dimensional variables (around 60 KPIs in our experiment), not to mention that the feedbacks of the RAN can be delayed. Third, in our study, we do not have access to the detailed settings of eNodeB provided by one of the largest wireless vendors. Instead, the vendor only provides several parameters to tune the devices. This makes the RAN a black box for us. Thus, training the model with a simulator is not applicable to our study. Fourth, unlike training with simulations, training in a real environment cannot be fast-forwarded. This slows down the training progress. Fifth, in many situations, the online control algorithm development involves the interaction with the network, which may influence the current customers. Offline study alone is usually not enough to acquire the long-term consequences of certain actions.

Several previous works have made some efforts to optimize the mobile network. [2] converts the experts’ knowledge into a rule-based configuration algorithm. [3] applies game theory to reduce the transmission power, where the Nash equilibrium can be achieved through a distributed recursive game. [4] proposes a biologically inspired spectrum sharing algorithm, which performs decentralized spectrum sharing and requires no inter-user coordination. [5] handles the handover problem by setting a fixed rule of the changing conditions. [6] considers three handover metrics: handover failure ratio, ping-pong handover ratio, call drop ratio. Authors minimize the weighted sum of these ratios over two tuning parameters, time-to-trigger and hysteresis by estimating the metrics functions of parameter, while assuming the metrics are independent. [7] combines both load balancing and handover algorithms with the consideration of the interaction of the two SON algorithms. [8] models the load balancing problem in explicitly parametric form, and further gets the optimal parameters.

More recent efforts begin to apply advanced machine learning techniques, such as reinforcement learning (RL). Because of the simplicity and high capability of the deep RL framework, it can be generalized to many control tasks. RL is a trial-and-error process, which can replace humans doing the same job but can outperform humans in many ways. In addition, the great supports from the third-party communities and many open-source off-the-shelf tools, such as TensorFlow [9], OpenAI gym [10], make the implementation practical and straightforward. The reinforcement learning framework is composed of several key components, state: which describes the status of the environment; reward: defined by users to pursue their goals. The goal of RL is not to maximize the immediate reward at each time step, but to increase the discounted sum of rewards in long term; agent: the agent needs to take an optimal action according to the state to achieve larger rewards in long...
term. For the details about the framework and state-of-the-art algorithms, we refer readers to [11], [12], [13], [14]. The RL frameworks have been applied on a large variety of network entities, such as the Internet of Things (IoT), Vehicular Ad hoc NETwork (VANET), Dynamic Adaptive Streaming over HTTP (DASH), Long Term Evolution (LTE) networks, 5G networks, etc. It has been used to let user devices select the channels to maximize the throughput [15], [16], [17], to reduce communication latency [18], [19], to reduce spectrum cost [20] or to design an optimal MAC scheduler [21]. RL can also be adapted to assign users to base station [22], to allocate network bandwidth [23] or to cache content [24], [25], [26]. Besides these works, [27], [28], [29], [30], [31], [32] apply RL techniques to increase the data rate and to enhance the quality of experience. [33], [34] provide good surveys in these topics. It is worth mentioning that almost all these studies are performed on simulators. To the best of our knowledge, we do not know there exits an eNodeB simulator that can handle thousands of eNodeB configurations and rules without oversimplifying the network functionality or losing flexibility.

There are many configurations for controlling various aspects of a RAN. For example, the media access control (MAC) scheduler allocates radio resources to mobile devices according to their radio frequency (RF) conditions and average throughput. Another RAN configuration is the handover setting. The network offloads traffic from overloaded cells to under-utilized cells to improve overall throughput by changing thresholds based on the observed traffic imbalance. Some others are power on/off, transmit power, tilt, etc. As the network conditions or traffic status change from place to place or from time to time, all these configurations need to be adjusted accordingly. This paper focuses on the MAC scheduler. The goal is to find an optimal policy for setting LTE eNodeB MAC scheduler configurations. The eNodeB scheduler allocates the available time-frequency radio resources to UEs at every transmission time interval of 1ms [35]. A UE can be a cellular telephone, a smartphone, a tablet computing device, a laptop computer, a pair of computing glasses, a wireless-enabled wristwatch, or any other cellular-capable mobile telephony and computing devices (broadly, “a mobile endpoint device”). The scheduling method significantly impacts the throughput of individual users, fairness between UEs, as well as throughput of the eNodeB cell [35]. A wise selection of the configuration option based on UEs’ status would improve the overall network throughput and provide benefits for speed-sensitive applications such as multimedia streaming [36], augmented reality and virtual reality games [37], video calls [38], etc. Popular scheduler strategies include round-robin, maximum C/I, proportional fair and etc [35], [39], [21]. These methods concern different aspects of balancing throughput, delay, fairness, and spectral efficiency.

We aim to build an intelligent controller to overcome the above challenges mentioned earlier. To achieve this goal, we first build a closed-loop control testbed RAN in the lab environment with one eNodeB, four UEs, and some other supporting servers. In this way, we can fully control the RAN without influencing the customer. Next, we introduce a deep reinforcement learning agent with data-driven and model-free algorithms to handle the high dimensional input and to learn a complex policy without strong assumptions about the environment. The model is trained by interacting with the RAN, including changing configurations of the eNodeB and getting the live feedbacks of the key performance indicators (KPIs) from the RAN. During the experiment, we monitor the status of both eNodeB KPIs and UEs status. We test the agent on selecting optimal eNodeB MAC scheduler settings. To the best of our knowledge, our work is the first one studying the RAN control problem using real apparatus. The rest of this paper is organized in the following order. Section II describes each platform module and the experiment paradigm in details. Section III gives the details about the RL algorithm. Section IV presents the results. We extend our ideas and concerns in section V. Finally, the paper is closed by the conclusion section VI.

II. Experiments

![Fig. 1: The framework of the RAN platform, which consists of one eNodeB, four UEs and some supporting servers. The reinforcement learning agent controls the configurations of the eNodeB.](image)

One of the main contributions of this work is that we set up a reinforcement learning testbed in a RAN test room, which consists of eNodeBs inside and completely blocks outside signals to affect the inner RF environment. We trained and tested the artificial intelligent agent on it. The diagram of the experiment platform is shown in Figure 1.

To automate the environment to implement live RL, we utilized the API provided by the vendor to change the eNodeB configurations remotely for action change. We collect the rewards/states with Kafka digesting eNodeB’s live call trace records. Besides, we use a laptop to control/schedule UEs’ behaviors to guarantee test repeatability. Meanwhile, we build Grafana dashboard to monitor the system online and trace history easily.

a) eNodeB: There are many sets of configurations for an eNodeB, including MAC scheduler, handover offsets, power
on/off, transmit power, sector tilt, remote radio heads, baseband units, etc. This work only focuses on one set, the MAC scheduler. Our vendor provides five options for the eNodeB MAC scheduler configuration. Each one corresponds to a different setting for handling resource allocation. The options range from "equal rate" to "maximize the UE throughput with the best RF". These options are used to balance the trade-off between throughput and fairness. Note that the design of the detailed settings is not provided by the vendor, thus rule-based scheduler configuration or training through simulator are not reliable and impractical due to the extreme complication.

b) User equipments: Four Samsung Android smartphones were used as UEs, they were set in USB tethering hotspot mode, and all applications were muted. Then they were tethered with laptops using USB cables, which controlled all the traffics. The upload/download traffic was generated by the laptops using iperf command tool provided by a leading company in wireless. The function is similar to the public perf network performance counter command tool, except that it is specialized for the cellular network owned by the wireless company. It guarantees the traffics go through the same link path and thus reduces the artifacts introduced by the disturbance of the network. All the traffics from the laptop went through the UEs. To create more realistic traffic, we analyzed the historical traffic logs of eNodeB measurement reports collected from a busy sample cell. This was because the scheduler played a more important role when the cell was handling multiple UEs simultaneously. Usually, an eNodeB might connect to many UEs at the same time within the service area, so we mimic this behavior with 4 UEs in our test. For the selected sample cell and time period, we got each data session’s traffic volume and RF condition by joining RRC traffic volume with reference signal received power (RSRP), reference signal received quality (RSRQ) measurement report. The RRC and RF conditions were learned from these recorded data. The k-mean clustering algorithm was performed on the data based on RF conditions and RRC sessions with four clusters. The clusters’ centers gave the means of RF conditions and RRCs. We calculated the variance of the RRC using the data from each cluster. Since the time dependency was small, the UEs’ RRCs were generated independently from Normal distributions with the estimated means and variances. Also, we adjusted the UEs’ RF conditions by changing the positions in the lab or covering them by Faraday bags. The RSRP values were -115dBm, -110dBm, -105dBm, and -94dBm respectively. Standalone tests were conducted to confirm the significant impact of RSRP values on the UE throughput in the lab.

c) Communication servers, key performance indicators, and monitor server: In our platform, a KPI composer/publisher server fetched the raw information from the eNodeB, then calculated the KPIs from the raw data and sent them to the reinforcement learning agent. The KPIs of the eNodeB were sampled every minute, such as the spectrum efficiency, cell throughput, number of active UEs, etc. If the sampling frequency was too high, then many data transmission package could not be finished. If the sampling frequency was too low, the status of the network was not well captured. The KPIs are selected to reflect the quality of the service. We followed the technical specifications (TS 32.450) there to calculate the values. The KPIs included information on both cell level and UE level. As for cell level KPIs, it included uplink and/or downlink volume/throughput, physical resource block (PRB), control channel element utilization (CCE), neighbor cells relations, handovers, frequency bandwidth, user geographic distribution, reference signal received power (RSRP), reference signal received quality (RSRQ), channel quality information distribution (CQI), timing advance distribution (TA), cell bitrate, cluster harmonic throughput, etc. As for UE level KPIs, it contained video user downlink throughput (video specific), radio frequency conditions (RF), number of active UEs, UE harmonic throughput, UE throughput gap (the difference between the maximum and minimum UE throughputs), worst UE throughput, etc. These KPIs would be later formatted into states or reward for the reinforcement learning agent. Then the agent updated the states and sent an optimal action to operate the eNodeB.

d) Reinforcement learning agent: In the closed-loop training, the RL agent needed the states of the RAN and the feedback reward as the input, and the actions operating the eNodeB as the outputs. The states were defined as part of the KPIs introduced above. The actions were set as eNodeB MAC scheduler options. The reward was set in two ways, one is the eNodeB overall throughput, the other one is the gap between UEs’ throughput, which was used for users’ fairness. The details of the RL algorithm will be discussed in section III. The reward can be defined in other ways for different purposes. We extend the discussion in section V.

e) Experiment paradigm: In one training episode, the pseudo-random users’ demands lasted for 80 minutes, followed by a 10 minutes rest, during which the RL agent was not trained. The demands episodes were repeated so that the evaluation would be easier than comparing the average performance obtained from different situations. This could eliminate the artifacts potentially caused by the users’ different demand patterns. Besides the training episodes, some baseline episodes were arranged by setting the policy with different constant actions. Two sets of experiments were carried out, one was to achieve better cell throughput, the other one was to minimize the gap of UEs’ throughput. For the first set of experiments, the baseline policy was the “maximize the UE throughput with the best RF” configuration. For the second set, the baseline policy was the “equal rate” configuration. The baseline policies agreed with our expert’s optimal design using domain knowledge.

III. REINFORCEMENT LEARNING MODEL

The radio resources allocated at any moment would affect all the active UEs’ sessions. When a session lasts long, it also has an impact on new coming UEs. Therefore, the eNodeB configuration change at any moment has a long term influence in the network. Thus, the scheduling process was modeled as a Markov decision process. In general, the goal of the RL is to find an optimal policy that can maximize the discounted sum of the rewards, not just the short-sighted immediate reward. In this way, the delayed reward can be taken into consideration.
This property makes RL attractive in many control problems. Meanwhile, finding such an optimal policy is a difficult task, especially with the high dimensional state space. The recent development of RL has made exciting progress by taking the advantage of the deep neural network over Q-learning [41].

We define the state $S_t$ at time $t$ using the KPIs received from the eNodeB, a vector of 58 entries. It contains the KPI values such as the spectrum efficiency, cell throughput, proportion of the utility, channel quality indicator (CQI) ratio, etc. The state does not include the information about individual UEs, as the framework is designed for general purpose, not relying on specific UEs. The action $A_t$ is discrete, which contains all possible eNodeB MAC scheduler configuration options "EQUAL_RATE", "PROPORTIONAL FAIR_HIGH", "PROPORTIONAL FAIR_MEDIUM", "PROPORTIONAL FAIR_LOW", "MAXIMUM C_OVER_1". $R_{t+1}$ is the reward received after taking action $A_t$ given the state $S_t$.

In our experiment, we aimed to improve the throughput for the long-term, thus we defined the reward as the spectrum efficiency. We also tried defining the reward as the gap between UEs’ throughput, i.e the minimum UE throughput minus the maximum UE throughput. This design encouraged fairness between UEs. To make the training more stable, we clip the reward and KPIs to a small range to avoid gradient explosion. $\gamma$ is the discount factor. If it is larger, the return considers longer-term of rewards. We fix it as 0.95. The goal of our model is to maximize the return $G_t$ as follows,

$$G_t = \sum_{i=t}^{\infty} \gamma^{i-t} R_{t+i}(S_t, A_t), \quad 0 < \gamma < 1$$

A neural network to approximate the state-action value function, which is known as the Q-function. The neural network has one hidden layer with 32 units.

$$Q(s, a; \theta) \approx \max_{\pi} \mathbb{E} [G_t | S_t = s, A_t = a, \pi]$$

Where $\pi$ is the policy, a mapping from states to action. With a good approximation of the Q-function, we select the optimal action given a state that maximizes the discounted sum of all future rewards. In each step, the action is taken according to $A_t = \arg \max_a Q(s_t, a; \theta)$ with probability $1 - \varepsilon$ or is selected randomly with probability $\varepsilon$. $\varepsilon$ balances the trade-off between exploration and exploitation. In the beginning, $\varepsilon = 1$ is large to encourage the agent to explore the unknown environment. $\varepsilon$ decays exponentially down to 0.1 so that the operation relies more on the trained model. Recent researches propose many heuristic methods to efficiently approximate the Q-function, such as the double Q-learning, prioritized memory replay, multi-step TD learning, etc. [42]. Here we implement the algorithm by combining several methods. The task is continuous instead of episodic, so there is no terminal state during the training. As in many offline learning algorithms [41], [43], [42], we train the model by replaying the experience. The replay experience buffer records the experience tuple $(S_t, A_t, R_{t+1}, S_{t+1})$ in time order. At each time step, the update rule for Q-function is the following. First, we randomly sample a batch of continuous segments of experience from the replay buffer, since we are using n-step TD learning.

$$\theta_{t+1} = \theta_t + \alpha \left( Y^{\text{DoubleQ}}_t(Q(S_t, A_t; \theta_t)) \cdot \nabla_{\theta} Q(S_t, A_t; \theta_t) \right)$$

$$Y^{\text{DoubleQ}}_t := R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} Q(S_{t+n}, A_{t+n}; \theta_t; \theta_t)$$

$$\hat{\theta}_{t+1} = (1 - \tau)\hat{\theta}_t + \tau \theta_{t+1}$$

Two neural networks are used with parameters by $\theta$ and $\hat{\theta}$ to alleviate the maximization bias or so called upward bias [44], [45]. After one iteration, we partially update the parameter $\theta$ into $\hat{\theta}$ to make the training more stable [43].

Working a the real apparatus will bring more persuasive results. On the other hand, we have to compromise on the fine-tuning as we can not fast forward a real process. We utilize existing eNodeB reporting network measurements to derive KPIs. Cell level measurements are reported periodically like every minute and UE level measurements are session-based. In our study, we build the KPI composer/publisher to aggregate KPIs once every minute.

IV. RESULTS

Figure 2 presents the results of 3 experiments. The first two are experiments for optimizing the cell throughput, where the reward is defined as the cell throughput. The third experiment is for optimizing the gap of UEs’ throughput, in which the reward is defined as the difference between the minimum UE’s throughput minus the maximal ones, the action is set as. The solid curve in Figure 2 is the mean reward of each episode, and the dotted curve shows the standard error band of the mean. The standard error band is used to verify whether the improvement is significant or simply due to the variance of the data. The dashed line represents the mean reward of all baseline episodes. In the first two experiments, the action is set as “EQUAL_RATE” as a constant. In the third experiment, the action is set as “MAXIMUM C_OVER_1” as a constant. Other actions are also tried, but the performance is not as good as these settings. In the beginning, the RL agent is not well trained so the performance is below the baseline. As the training goes by, the performance elevates gradually and finally outperforms the baseline. The constant baseline policies can achieve fair performance, but to get further improvement, the model should take the dynamics of the system into the consideration, which is hard for a model designed with only domain knowledge.
V. DISCUSSION

To improve the performance of the RL model and to transfer the algorithm from the testbed to a massive network we need to consider the following inevitable challenges.

a) Training and tuning: Because the sampling rate of the platform was slow, we had to compromise on the thorough training and fine-tuning. This was the cost of dealing with the actual environment. Limited by these issues, we are not clear about the ceiling of the performance, but at least the model has the potential to do better than our results with longer experiments. In practice, algorithm debugging is another problem with this slow responding testbed. So, we developed and tested the software framework in the OpenAI gym with coarse tuning to verify if the reward curve could boost up quickly in the first few episodes. We noticed the empirical clue that a model with a larger stable reward tends to have faster training in the first few episodes, so we pay more attention to the beginning sessions. Usually, a smaller model needs fewer data to train, so we chose simpler algorithms and smaller neural networks. But the model might be not rich enough to capture more complicated optimal policy. To get faster training, we tried prioritized experience replay, however, due to the noisy data, this method does not help too much. The large variance contaminated the weighted experience sampling, so it was hard to tell whether a sample had a large TD error representing or with large noise. Besides these efforts, we tried the pre-training method using historical records by following the method in \[46\], \[47\]. However, the historical data were not from an expert’s demonstration, so the pre-training does not yield significantly better results. This was also seen in our OpenAI gym test experiments with only historical data but not experts guidance. A promising method to train the model faster is to parallelize the process with more platforms running simultaneously and asynchronously, such as \[48\], \[49\], \[50\]. Except for these endeavors, we still do not have a good strategy for model tuning. We argue that this is not just an issue for our experimental design, but a limitation in general for reinforcement learning or optimal control dealing with the complicated real environment.

b) Safety: RL training is notorious for its instability, that the reward curve can cliff fall unpredictably. If such a model is deployed in the RAN with millions of customers, the consequence can be catastrophic. This potential high risk leads to the conservativeness of the application in industry. The RL agent is designed to maximize the reward in long term, so the dropping down of the reward in a few steps is not prohibited. In addition, the risk is related to the uncertainty of the environment, so even an optimal policy can perform poorly in some situations. Many exploration methods by default are blind to these risks of actions. Maximizing the reward in long term does not guarantee positive outcomes in every step. On the other hand, without exploring in the real environment, the model will hardly get a chance to exploit better. This concern has been addressed in the safety reinforcement learning. See a good survey \[51\] and the references therein. For our case, the strategies can be, 1. train the model first then transfer it to the production without changing; 2. gradually exploit the model from regional to global; 3. prohibit several exploration options based on domain knowledge or historical records.

c) Customization: While the reinforcement learning framework is very general for complicated environments,
applying it to a wide range of problems still requires a deep understanding of the domain. A critical assumption of RL is that the underlying Markov decision process does not change or changes very slowly. The behavior of the testbed may be very different from that in the real network due to the spatial and temporal heterogeneity. So, patching the issues still requires further understanding and deep domain knowledge.

d) Combination of configurations: In section II paragraph eNodeB, we list many aspects of the configurations for the eNodeB, but we only provide experiments for the MAC scheduler. It is interesting to combine several configurations to achieve multiple goals. For example, if we combine the MAC scheduler and power on/off, the goal can be maximizing the cell throughput while using the minimum amount of energy. A natural question is that if the RL agents were trained for MAC scheduler and power on/off separately, and the two sub-agents share some parameters, how to merge them into a new one with multiple goals. This saves a lot of time from re-training the model from scratch. Another problem is that if the selected action from the two agents is different, then how to solve the conflicts.

VI. CONCLUSION

In this work, we first set up a RAN testbed in a lab environment for SON optimal policy design. Then we exploit recent advances of artificial intelligence and implement a deep reinforcement learning algorithm on the real environment and get satisfactory improvement for the complicated dynamic RAN. After stepping out of the academic RL research which heavily relies on simulators, we find more challenges such as lack of training data, model tuning, model scale-up, etc. Our work serves as a stepping stone for the following studies dealing with the real environment.

REFERENCES

[1] J. Ramiro and K. Hamied, Self-organizing networks: self-planning, self-optimization and self-healing for GSM, UMTS and LTE. John Wiley & Sons, 2011.
[2] K. Unemoto, T. Imamura, and G. Takahashi, "Radio access network design and optimization technology," NEC Technical Journal, vol. 1, no. 2, pp. 56–60, 2006.
[3] F. Wang, O. Younis, and M. Krunz, "Gmac: A game-theoretic mac protocol for mobile ad hoc networks," in 2006 4th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks. IEEE, 2006, pp. 1–9.
[4] B. Atakan and O. B. Akan, "Biologically-inspired spectrum sharing in cognitive radio networks," in 2007 IEEE Wireless Communications and Networking Conference. IEEE, 2007, pp. 43–48.
[5] A. Lobinger, S. Stefanski, T. Jansen, and I.-M. Balan, "Load balancing in downlink lte self-optimizing networks," in 2010 IEEE Vehicular Technology Conference (VTC-Spring). IEEE, 2010.
[6] T. Jansen, I. Balan, J. Turk, I. Moerman, and T. Kurner, "Handover parameter optimization in lte self-organizing networks," in 2010 IEEE 72nd Vehicular Technology Conference-Fall. IEEE, 2010, pp. 1–5.
[7] A. Lobinger, S. Stefanski, T. Jansen, and I. Balan, "Coordinating handover parameter optimization and load balancing in lte self-optimizing networks," in 2011 IEEE 73rd vehicular technology conference (VTC-Spring). IEEE, 2011, pp. 1–5.
[8] H. Hu, J. Zhang, X. Zheng, Y. Yang, and P. Wu, "Self-configuration and self-optimization for lte networks," IEEE Communications Magazine, vol. 48, no. 2, pp. 94–100, 2010.
[9] "Tensorflow," [https://www.tensorflow.org/], accessed: 2019-05-20.
[10] "Openai gym," [https://gym.openai.com/], accessed: 2019-05-20.
[11] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.
[34] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, “Applications of deep reinforcement learning in communications and networking: A survey,” IEEE Communications Surveys & Tutorials, 2019.

[35] M. T. Kawser, H. Farid, A. R. Hasin, A. M. Sadik, and I. K. Razu, “Performance comparison between round robin and proportional fair scheduling methods for lte,” International Journal of Information and Electronics Engineering, vol. 2, no. 5, pp. 678–681, 2012.

[36] A. Ahmedin, A. Ghosh, and D. Ghosal, “A survey of multimedia streaming in lte cellular networks,” arXiv preprint [arXiv:1803.05080] 2018.

[37] Y. C. Sung, Y.-S. Ho, Y.-B. Lin, J.-C. Chen, and H. C.-H. Rao, “Voice/video quality measurement for lte services,” IEEE Wireless Communications, vol. 25, no. 4, pp. 96–103, 2018.

[38] A. Saxena and R. Sindal, “Performance analysis of mac scheduler in lte (eutran) for asar”: Resource allocation,” in 2016 10th International Conference on Next Generation Mobile Applications, Security and Technologies (NGMAST). IEEE, 2016, pp. 6–11.

[39] C. Westphal, “Challenges in networking to support augmented reality and virtual reality,” IEEE ICNC, 2017.

[40] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in International conference on machine learning, 2016, pp. 1928–1937.

[41] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al., “Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures,” arXiv preprint [arXiv:1802.01561] 2018.

[42] L. Espeholt, R. Marion, P. Staniczyk, K. Wang, and M. Michalski, “Seed rl: Scalable and efficient deep-rl with accelerated central inference,” arXiv preprint [arXiv:1910.06591] 2019.

[43] J. Garcıa and F. Fernández, “A comprehensive survey on safe reinforcement learning,” Journal of Machine Learning Research, vol. 16, no. 1, pp. 1437–1480, 2015.