ICRAN: Intelligent Control for Self-Driving RAN Based on Deep Reinforcement Learning

Azza H. Ahmed, *Member, IEEE*, and Ahmed Elmokashfi

***Abstract*—Mobile networks are increasingly expected to support use cases with diverse performance expectations at a very high level of reliability. These expectations imply the need for approaches that timely detect and correct performance problems. However, current approaches often focus on optimizing a single performance metric. Here, we aim to address this gap by proposing a novel control framework that maximizes radio resources utilization and minimizes performance degradation in the most challenging part of cellular architecture that is the radio access network (RAN). We devise a method called Intelligent Control for Self-driving RAN (ICRAN) which involves two deep reinforcement learning based approaches that control the RAN in a centralized and a distributed way, respectively. ICRAN defines a dual-objective optimization goals that are achieved through a set of diverse control actions. Using extensive discrete event simulations, we confirm that ICRAN succeeds in achieving its design goals, showing a greater edge over competing approaches. We believe that ICRAN is implementable and can serve as an important point on the way to realizing self-driving mobile networks.**

***Index Terms*—Self-driving network, slicing, RAN, resource allocation, performance optimization, deep reinforcement learning, ns-3 simulation, DDPG.**

I. Introduction

The FIFTH generation mobile network, 5G, has transformed the mobile network into a multi-service architecture that supports diverse use cases with varying requirements [1]. 5G virtualizes network resources and chains them into end-to-end network slices that are adapted to use cases' requirements. This flexibility makes mobile networks increasingly complex to manage [2]. Furthermore, several envisioned use cases like public safety communication and industrial control have stringent performance expectations. Hence, multi-slice 5G and 6G networks must be capable of quickly detecting and correcting performance degradation. Current mobile networks resort to pre-configured priorities, over-provisioning and at best implementing traditional closed loop control systems with a limited scope like in the case of self-organizing networks (SON) [3]. The need for intelligent automation has motivated the academia and industry to argue for building "autonomous" or "self-driving" networks, where network management and control decisions are made in real-time and in an automated fashion [4]. For instance, the Open Radio Access Network (O-RAN) architecture, which aims to realize the RAN as a set of visual network functions on commodity hardware, has identified supporting intelligent RAN control as a key design goal [5]. Despite all these effort, building "self-driving" mobile networks that are practically deployable has largely remained unrealized.

A major challenge in this respect, having in mind the scale of mobile networks, involves devising a control architecture that balances complexity and overhead. Further, there is a lack of unified control approaches that can deliver on multiple objectives (e.g., maximize resource utilization while ensuring an acceptable level of performance). More specifically, the existing proposals focused on tracking and optimizing a single metric like coverage [6], power management [7], throughput [8] and resource sharing [9] at a time. It flows directly from this limited focus that current approaches resort to often applying a single control action, e.g., adjusting base station transmit power. However, a multi-slice network is by definition a multi-service network. Hence, tracking a single metric is bound to assure quality for a subset of the services that run on the network. A viable approach to realizing self-driving mobile networks must be able assure quality for all running slices according to their priority and service level agreements (SLAs). Achieving this requires choosing and mixing diverse control actions, e.g., simultaneously adjusting coverage and optimizing resource allocation. Here, we aim to bridge this gap by proposing a machine learning based approach to manage resource utilization and performance in the RAN. We focus on the RAN, because its performance and reliability heavily impact users' quality of experience [10]. Our approach tracks and optimizes diverse use cases. To this end, it employs several different control actions.

We propose, ICRAN, an intelligent control scheme for a multi-slice RAN. ICRAN leverages deep reinforcement learning (DRL) to derive strategies for maximizing resource utilization and minimizing SLA violations under different network conditions. Reinforcement learning is a machine learning (ML) approach where intelligent agent/agents interpret and interact with their environment and learn how to achieve certain objectives via cycles of trial and error based learning. We introduce both a centralized and distributed control schemes. In the former, a single controller optimizes the configurations of all base stations, while each base station has own controller in the

Manuscript received 4 May 2022; revised 4 July 2022; accepted 13 July 2022. Date of publication 18 July 2022; date of current version 12 October 2022. The associate editor coordinating the review of this article and approving it for publication was M. Tornatore. *(Corresponding author: Azza H. Ahmed.)*

Azza H. Ahmed is with the Center for Resilient Networks and Applications, Simula Metropolitan Center for Digital Engineering, 0167 Oslo, Norway, and also with the Center for Resilient Networks and Applications, Oslo Metropolitan University, 0176 Oslo, Norway (e-mail: azza@simula.no).

Ahmed Elmokashfi is with the Center for Resilient Networks and Applications, Oslo Metropolitan University, 0176 Oslo, Norway.

Digital Object Identifier 10.1109/TNSM.2022.3191746

latter. Following an initial training phase, the controller continuously monitors the state of the network and immediately reacts to performance degradation by executing actions that extend coverage and regulate resource usage. We implement ICRAN using the OpenAI Gym framework [11] and the ns-3 simulator [12]. Our evaluations show that ICRAN converges quickly to strategies that help maximizing radio resource utilization and minimizing SLA violations for the entire network. ICRAN outperforms approaches that leverage DRL, implement adaptive priority-based resource management as well as those resorting to heuristics to react to network changes. The benefit from ICRAN spans regimes where the network is lightly loaded, running at its capacity, and is heavily loaded. For example, ICRAN utilizes 97% of available radio resources when the network is loaded at 200% that is 7% higher than the next best method. At the same time, it reduces the number of SLA violations for slices with stringent requirements, that is achieved by the next best method, by up to a factor of three. This work is the first to apply deep reinforcement learning to collectively solving multiple RAN control problems. The previous work has mainly focused on optimizing for a single control problem.

A. Contributions

The main contributions of this paper are summarized in the following:

1) We propose ICRAN a novel control framework based on DRL that is capable of maximizing resource utilization and minimizing SLA violations in a multi-slice RAN. ICRAN introduces a novel reward function and an action space with diverse actions which allows for optimizing multiple objectives collectively, namely antenna tilt, traffic load balancing, and resource allocation.
2) We investigate two different control architectures for our framework; centralized and distributed control. In the centralized framework, we formulate the problem as a single-agent DRL whereas, the distributed problem is solved via a multi-agent DRL.
3) Finally, we validate the performance of our proposed framework through extensive simulations using ns-3. The experiment results show that our method outperforms the state-of-the-art methods in radio resources management. Moreover, the advantage of our method persists under different networking conditions such as high congestion and radio failure.

B. Paper Structure

The remainder of the paper is organized as follows. The next section gives the background of this work, while Section III covers related research on RAN slicing and the application of DRL in mobile networks. In Section IV, we discuss the overall system model from a high-level perspective. We describe in Section V the scheduling algorithm with slicing constraints that we consider. Then, we proceed to elaborate our DRL problem formulation and the algorithms we develop to solve the problem. We present our experimental setup in Section VI and we evaluate the performance of ICRAN and compare it with some baselines and state-of-the-art methods in Section VII. In Section VIII, we discuss our findings, the implementation considerations and the limitations of this work and how to address them in the future. Finally, a brief concluding remarks are provided in Section IX.

II. Background

Unlike the previous generations of mobile networks, 5G is designed to support a wide range of services with diverse requirements. These requirements span a wide range in terms of expected throughout, latency, reliability, mobility and number of devices [13]. A single static network architecture, however, can not cater for this diversity. This motivated a pivot towards realizing multiple isolated logical networks, with different configurations, over the same physical infrastructure, i.e., network slicing [14], [15], [16], [17]. Recent advances in defining network elements and network configurations in software have made this pivot possible. Key technologies in this respect are software defined networking (SDN), network function virtualization (NFV) and cloud-native network functions (CNFs). SDN decouples the network data and control planes and centralizes network control which allows for a full network programmability [18]. NFV transfers network functions like routers and firewalls from specialized physical implementations (i.e., a hardware box with tailored software) to software implementations that can run as virtual machines on a general purpose computers. VNFs can be chained to form an end-to-end network architecture [19]. Finally, CNF is a natural evolution of VNF that shrinks them to run as containers and optimizes them to run in the cloud [20]. SDN and NFV/CNF complement each other towards building an end-to-end software defined and programmable network architectures. In the Long-Term Evolution (LTE) mobile core network, the data and control plane functions are realized by dedicated hardware that implements each specialized function. However, the 5G core is designed to be "cloud-native", in the sense that the functions that handle the control and data planes, e.g., UPF and AMF, could be deployed as VNFs/containers on a cloud infrastructure. The use of NFV and SDN has been started from core and networking middleboxes and extended to RAN functions. The development of open and intelligent RAN (O-RAN) has received great attention [21]. O-RAN is proposed to enhance the RAN performance through virtualized network elements and open interfaces that incorporate intelligence into the RAN. O-RAN introduces programmable components that can run optimization routines with closed-loop control and orchestrate the RAN. Specifically, the O-RAN has logical controllers that monitor the status of the network (e.g., number of users, load, throughput, resource utilization) and process this data leveraging AI/ML algorithms to determine and apply control policies and actions on the RAN, for example, network and RAN slicing, load balancing, handovers and scheduling [5].

III. Related Work

A. RAN Slicing

End-to-end slicing involves virtualizing the RAN, the transport and the core. The former is specific to mobile networks while the latter two are common across different types of networks, e.g., data centers. Further, RAN slicing requires an efficient approach to resource management given the limited nature of the virtualized-resource in question, i.e., radio spectrum. To this end, there are several proposals which we discuss next.

Various traditional algorithms have been proposed for tackling the resource scheduling and allocation problems in the RAN. Nojima *et al.* [22] presented three resource allocation methods for RAN slicing by slightly modifying the conventional MAC scheduling algorithms: 1) a static allocation method, which allocates Resource Blocks (RBs), that is the smallest units of resources in frequency and time that can be allocated to a user, in a fixed manner regardless of the channel conditions of users in each slice, 2) a round-robin allocation algorithm, which allocates RBs to each slice sequentially and based on the channel conditions of users in each slice, 3) a per-user priority algorithm that allocates RBs to users based on their priority within their respective slice. Their experiments showed that the priority-based algorithm achieved higher throughput compared to that resulted by the static and round-robin algorithms. However, all these methods do not consider satisfaction of the slice requirements when allocating RBs to a slice. Thus, Shrivastava *et al.* [23] proposed a method that allocates RBs to slices taking into account the desired SLA. This approach allows for a flexible assignment of RBs, which allocates temporarily unused RBs to slices in need with the possibility of allocating more RBs than necessary. To address this, Bakri *et al.* [24] proposed a data-driven mechanism for sharing RAN physical resources among different network slices based on optimal value for RBs. The proposed algorithm calculates the optimal value for the radio resources based on the Channel Quality Indicator (CQI) reports collected by the base stations. The algorithm is running at the slice orchestrator level which adapts to two slices with different quality of service (QoS) requirements, specifically ultra reliable and low latency communications (URLLC) slice and enhanced Mobile Broadband (eMBB) slice. To reduce the overhead of sending the CQI values between the base station and the slice orchestrator, the authors presented a machine learning model to predict the user equipment (UE) channel stability. If the channel quality is relatively stable, the CQI values do not vary much in time. Therefore, frequent CQI reports will not affect the performance of the slicing algorithm. Unlike our work, the proposed method calculates the required radio resources based on only one key performance indicator (KPI), i.e., either latency or throughput and thus cannot directly support other types of slices. Moreover, their results showed that there is a threshold on the number of users in each slice when the SLA is violated without considering how to minimize those SLA violations.

Aligned with the recent advances in DRL, Mei *et al.* [8] presented a hierarchical framework based on integrating the deep deterministic policy gradient (DDPG) and double deep-Q-network algorithm to solve RAN slicing problem. Specifically, this framework consists of two controllers: an upper-level controller which adjusts the slice configuration to improve QoS performance at a coarse granularity and a lower-level controller that schedules network resources and power allocation to active UEs in each network slice at a fine granularity. Their results showed that RBs in RAN slices can be managed efficiently using the DDPG for RB allocation. However, in this method, if the number of slices is different from the number of slices during training, RB allocation to slices is impossible. Therefore, RB allocation independent of the number of slices was proposed in [25], where Liu *et al.* presented a method called DeepSlicing which tackles the problem of resource allocation in multi-slicing networks in two stages. The first stage allocates resources to users within a slice through DRL that learns the optimal policy in each network slice to maximize the overall utilities of users in the slice while satisfying the users' SLA. The second stage coordinates the resource allocation across the network slices. Similar to our work, this work leverages the DRL advances in RAN resource management. However, our work goes beyond the efficient resources allocation solution to explore a large action space containing other possible actions such as antenna tilt optimization and load balancing between the base station to achieve near-zero violations of slices' SLA. To address the problem of the long training time needed by DRL methods, Abouaomar *et al.* [26] proposed a federated DRL mechanism to collaboratively train a DRL model for bandwidth allocation in RAN slicing. Their simulation results have shown that the model trained using federated learning is more robust against environment changes compared to models trained separately by each mobile virtual network operator.

B. Deep Reinforcement Learning in Mobile Networks

The advances in DRL have led to outstanding success in various domains. DRL has been recently proposed for solving many wireless communication problems. Several surveys summarized these works. For example, Luong *et al.* [27] provided a comprehensive overview of deep reinforcement learning application in communications and networking such as dynamic network access, data rate control, wireless caching, data offloading, network security and connectivity preservation. Compared to [27] which focused on single agent problems, Feriani and Hossain [28] presented an overview of both single-agent and multi-agent reinforcement learning as key enabling technologies of future wireless networks. They highlighted the potential for applying cooperative multi-agent reinforcement learning to different domains such as mobile edge computing (MEC), unmanned aerial vehicles (UAV) networks and massive MIMO. Other surveys reviewed the application of deep reinforcement learning algorithms in specific domains such as Internet of Things (IoT) [29], URLLC in 6G networks [30], vehicular networks in 6G [31] and mobile edge caching [32]. Recently, Seid *et al.* [33] proposed a multi-UAV enabled IoT edge approach for dynamic task offloading and resource allocation leveraging multi-agent DRL methods. They aimed to minimize the overall network computation cost while ensuring the QoS requirements of IoT devices or UEs in the IoT network.

In general, the previous proposals can be divided into two groups based on the action space. The first applies deep

Q-learning to problems with a discrete action space, while the second applies actor-critic methods to problems with a continuous action space. Further, almost all existing work revolves around executing a single principal action. For example, Nasir and Guo [7] proposed a power allocation scheme based on deep Q-learning model (DQN). Each transmitter collects channel state information and QoS information from several neighbors and adapts its own transmit power accordingly. Also, Li *et al.* [34] used DQN method to tackle the resource allocation multi-user computation offloading in wireless MEC. Recently, Ren *et al.* [35] addressed the problem of dynamic resource allocation for MEC slicing system using DDPG algorithm. They formulated the resource allocation problem as Markov decision process (MDP) in which, the wireless resources and computing resources are configured dynamically according to the requirements of different types of slices to maximize the network operator revenue. Similarly, Seid *et al.* [36] leveraged DDPG algorithm to optimize the computational costs and resource allocation to satisfy the QoS of Edge IoT devices. They proposed a collaborative framework where each agent learns from the previous offloading experiences and dynamically associates the nearest computational node with the UAV network. We refer the reader to the aforementioned surveys and the references for a comprehensive overview of previous efforts.

In this paper, we propose ICRAN as a control approach for performance optimization based on DRL in a multi-slice RAN. We present two variants of ICRAN: single agent and multi-agent. ICRAN uses an actor-critic method due to its continuous action space. This work is, to the best of our knowledge, the first work to date that presents a DRL-based approach that both maximizes radio resource utilization and minimizes SLA violations simultaneously. Also, our work is the first, in this application area, to propose the use of three inherently different categories of actions.

IV. System Model

In this section, we present the problem statement, the system and traffic models, and our control approach.

A. Problem Definition and System Architecture

Network performance optimization represents one of the major challenges for mobile network operators, especially in RAN [37]. Near-future mobile networks are going to support users with different service requirements by leveraging network slicing. Therefore, an efficient scheduling mechanism is essential for allocating available resources to these different types of services. Since traffic is dynamic, a static resource allocation is bound to fall short in maximizing resource utilization while maintaining the SLA. Hence, there is a need for an adaptive control mechanism that steers the network in response to traffic demand variations and anomalies. This paper proposes a DRL-based framework, *ICRAN*, that maximizes resource utilization and minimizes SLA violations in a multi-slice RAN.

We investigate RAN control in LTE cellular network consisting of M eNBs that serve N users, who connect to three different slices with diverse service requirements (see Figure 1). The number of users that connect to the three slices is N_{S1}, N_{S2} and N_{S3}, respectively. The core network contains the serving gateway (SGW), packet data network (PDN) gateway (PGW), and the mobility management entity (MME). Among other functions, the MME is responsible for paging procedures of UEs upon arrival. When a new UE arrives to the network, a logical channel is established between the UE and eNB called a radio bearer, which also connects the UE to IP-based networks through the evolved packet core (EPC). The radio bearer is associated with QoS parameters based on the application it serves. A key parameter in the respect is the QoS Class Identifier (QCI).

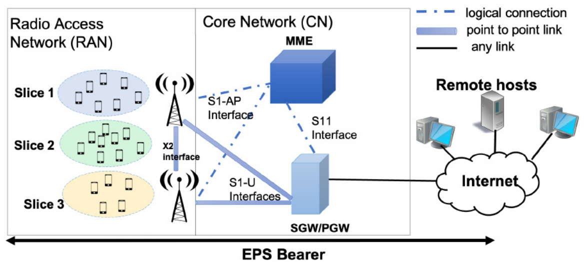


Fig. 1. The LTE reference architecture that we use in our framework.

We choose to focus on LTE because realistic 5G simulation models are still under development. For example, the current models do not support handovers yet [38], which is an essential feature in mobile networks and in our framework. Nevertheless, we believe that this choice has no impact on the generalizability of our results to 5G networks, because we are not making any assumption that is only limited to LTE. Our simulation implements all the 4G core network components (PGW, SGW, PDN, MME and EPC), UE, MAC layer, as well as all higher protocol such as Radio Link Control (RLC), Packet Data Convergence Protocol (PDCP), Radio Resource Control (RRC) which closely resemble that of 5G. Unlike 4G, 5G base stations (i.e., gNBs) and UEs might support multiple numerologies, however this feature does not affect the functionality of ICRAN.

In our system model, we assume a DownLink (DL) dominated traffic model, in agreement with today's traffic patterns [39]. Generally speaking, LTE supports two traffic patterns:

- Guaranteed Bit-Rate (GBR): GBR traffic requires a constant throughput, irrespective of the required resources it takes to fulfill it. Voice-over-IP (VoIP) is an example of an application that expects a GBR.
- Non-guaranteed Bit-Rate (Non-GBR): this traffic pattern does not have rigid throughput requirements and can use any unused resources left by guaranteed services. Applications like Web browsing can be considered as a non-GBR service. Based on the users' channel conditions and scheduler decision, the throughput is determined.

LTE introduced several QCIs to tag different traffic patterns, which have been used by various works to realize the slicing [40], [41], [42]. Here, we leverage QCI values to define three slices that we summarize in Table I. The first slice represents the GBR service which has the strictest SLA both in

TABLE I
SLICES DEFINITION

Slice	Application	QCI	Resource Type	Priority Level	Delay Budget	Packet Loss Rate	Minimum UE Throughput(Kbps)
1	VoIP	1	GBR	2	$100ms$	$10.E-2$	64
2	Video	7	non-GBR	6	$300ms$	$10.E-6$	525
3	Best Effort	9	non-GBR	9	$300ms$	$10.E-6$	N/A

terms of throughput and delay requirements. Both *Slice 2* and *Slice 3* are non-GBR services, however, *Slice 2* has a minimum throughput that the network should provide. Finally, *Slice 3* is the best-effort slice that uses the remaining resources. We assume that every user is a member of only one of the three slices. This is a reasonable assumption for the majority of users.

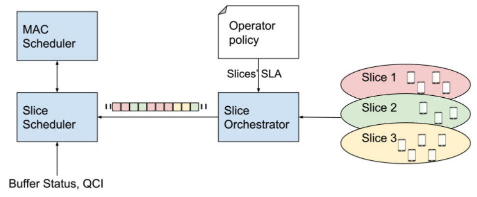


Fig. 2. Our RAN slicing model.

B. Control Frameworks and Levers

We propose two different control architectures for ICRAN.

1) Centralized ICRAN (ICRAN-C): In this architecture, we consider a single centralized controller (agent) that can fully observe the network information and reconfigure the entire network accordingly. Even though a centralized decision-making may be the best in performance, it is usually impractical due to the potential signaling overhead. Furthermore, in practical implementations, a centralized solution can be slow. It is difficult for a centralized algorithm to quickly find an optimal solution because the search space increases exponentially as the size of the network increases [43].

2) Distributed ICRAN (ICRAN-D): To avoid the pitfalls of centralized control, a partially centralized or a fully decentralized control architecture is needed. Here, each eNB takes its own decisions without considering other eNBs' state and decision, or with little information about all or some eNBs, for example neighbouring eNBs. Having a fully independent decentralized architecture, may result in conflicting strategies as each eNB is essentially trying to find its own optimal solution [43]. Therefore, in this architecture we assume the existence of a communication channel between the eNBs to exchange the information needed to find the optimal solution. We can utilize the existing X2 interface (or Xn interface in 5G stand-alone) between the eNBs (or gNBs in 5G) to exchange information needed for RAN control.

As control is essentially the act of adapting the state of a system in response to internal and external stimuli, a controller needs levers to adjust the state of the system. Here, we exploit three primitives that are available in today's networks, which we describe next.

1) Optimize antenna tilt: There are some parameters that can be used to optimize the network coverage and capacity. Antenna tilt is one of these parameters that can be easily modified in an automatic way in order to optimize the network coverage. Antenna tilt can be defined as the inclination angle between the antenna's main beam and the horizontal plane [6]. The optimal antenna tilt value for a cell depends on the tilt values of its neighbors; too much downtilt can result in coverage holes, while too little downtilt will lead to interference with neighboring cells. We assume that only one cell can update its antenna tilt at each time step. This makes it easier to identify the impact of that change on coverage and performance.
2) Performance triggered handovers: In conventional LTE networks, handovers are mainly event-triggered. UEs measure their signal quality and report it back to the serving eNB, which uses these reports to initiate handovers when needed. Besides this, we redefine handover to include performance triggered handovers. For instance, the intelligent controller can shift users to a nearby eNB to enhance their performance. If the serving eNB is becoming overloaded, some UEs, despite good coverage, will need to switch to other eNBs for a better service and a lower delay.
3) Optimize EPS bearer rate: Evolved Packet System (EPS) bearer is defined with certain data rate according to the QoS parameters: GBR, Maximum Bit Rate (MBR), Aggregated Maximum Bit Rate (AMBR) and QCI. For GBR type applications (e.g., VoIP) which requires a constant data rate, the data rate is controlled via the GBR parameter. For non-GBR applications, such as video and best effort Internet (BE), which require a variable amount of bandwidth, the traffic is controlled by the aggregated maximum data rate (AMBR). Utilizing these data rates parameters, we can optimize the resources allocated in terms of deciding the optimum bearer rates for different users dynamically according to their QoS requirements.

V. APPROACH TO INTELLIGENT RAN CONTROL

In this section, we present the underlying network slicing scheme, the formulation of ICRAN as a deep reinforcement learning problem and its two architectures: ICRAN-C and ICRAN-D.

A. RAN Slicing Model

The allocation of radio resources to slices according to their requirements is a fundamental part of network slicing that is usually executed at RAN. In line with the works [22], [23] discussed in Section III-A, we build our adaptive RAN slicing scheme (see Figure 2) by modifying the existing MAC scheduler and implementing a hierarchy of schedulers. Note that the LTE MAC scheduler is responsible for allocating radio resources to UEs. The frame structure of the downlink air interface contains ten subframes of $1ms$ each, which is also the transmission time interval (TTI). Besides the slice scheduler, our slicing model comprises a slicing orchestrator that determines the assignment of RBs to slices and instruct the scheduler to execute that based on the operator policy. It informs the scheduler of which slice to schedule at the current subframe based on the QCI value. If the scheduled slice has no data to transfer, the slice with the highest priority and data waiting to be transferred will be scheduled. The slice scheduler determines the slice priority based on the QoS constraints provided by the EPS bearer, which is associated with the QCI. Furthermore, the slice scheduler receives a set containing the buffer status of all UEs for each slice to determine the amount of data to be transferred. Within each slice, RBs are assigned to UEs using a conventional MAC scheduler that implements the proportional fair scheduling algorithm [44]. Our control algorithm extends this slicing scheme by adding a layer that manipulates coverage and resource allocation in response to network dynamics and performance degradation.

B. DRL Problem Formulation

We use deep reinforcement learning as a candidate solution for our problem, because DRL can accommodate the complexity of network dynamics. The future mobile networks are large-scale and complex in the sense of supporting diverse use cases which results in large state and action spaces, and the conventional control methods may not be able to find the optimal decision in reasonable time. Thus, we present the ICRAN framework based on DRL environment that comprises the real-time state of all eNBs and UEs in our slicing network. The ICRAN agent (or a set of agents) monitors the status of the eNBs including the attached UEs and network performance indicators. Based on the obtained information, the agent makes a decision whether to increase the antenna coverage, distribute the traffic loads or adjust the data rate for a specific slice to maximize the overall network performance and minimize SLA violations. The decisions are made at each control interval. The value of the control interval depends on time needed to execute the actions in the environment. The environment returns a reward, which is calculated depending on the QoS and the network efficiency, and then a new decision process will be activated. In this work, we formulate this problem in two different architectures based on the control level: 1) single agent DRL (i.e., centralized control) and 2) multi-agent DRL (i.e., decentralized control).

Formally, our centralized control problem is MDP, which is modeled as a 4-tuple $\langle S, A, r, P\rangle$, where S is the state space, A denotes the action space, r represents the reward function and P is the state transition probability for state s and action a. The state space involves the state of M eNBs and all users connected to them. The agent relies on the policy π to select actions for RAN control to maximize the reward. The task of DRL is to find the optimal policy $\pi^* : S \rightarrow P(A)$ that maximize the expected return. The return from a state s is defined as the sum of discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$ with a discounting factor $\gamma \in [0, 1]$. A deterministic policy returns actions to be taken in each perceived state, while the stochastic policy returns a distribution over actions. We define the Q-function, which measures the expected accumulated rewards under policy π as shown below:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_i \geq t, s_i > t \sim \mathbb{E}, a_i > t \sim \pi}[R_t \mid s_t, a_t] \quad (1)$$

According to the Bellman equation, the relation between the Q-function and the immediate reward can be formulated as:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim \mathbb{E}}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]] \quad (2)$$

We elaborate on the algorithm we choose to maximize this Q-function later in this sequel.

For the decentralized architecture, we extend the MDP formulation to the multi-agent setting taking into account the communication between agents [45]. We define our problem as a partially observable Markov game (POMG) for M agents comprising of a set of states S that contains the possible information of all agents, a set of actions A and a set of observations O for each agent. At each time step t, each agent takes an action $a(t)$ based on state $s(t)$, moves to new state $s(t + 1)$, receives a new observation $o(t + 1)$, and finally receives an immediate reward r. Similar to the MDP model, the agent in POMG also aims to find the optimal policy in order to maximize its expected long-term discounted reward.

Next, we elaborate on the details of our formulation.

1) State Space: The RAN controller works as an agent interacting with the network environment at every time step t. The state of the i-th eNB at time step t, $s_i(t)$, is given by:

$$s_i(t) = [DT(t), Th(t), UE_{S1}(t), UE_{S2}(t), UE_{S3}(t)] \quad (3)$$

where $DT(t)$, $Th(t)$ are the antenna downtilt and average throughput for the eNB respectively. $UE_{S1}, UE_{S2}, UE_{S3}$ are three vectors representing the identifiers for the UEs attached to eNB from three slices S_1, S_2, S_3. For each UE in one of these three list, we use the Flow Monitor module in ns-3 to track its throughput and end-to-end delay at each time step.

2) Action Space: The action space is a vector A representing our control levers: antenna tilt optimization, traffic load balancing and traffic shaping.

- Antenna tilt optimization. For each antenna this action is defined by three discrete variables: $[a_{-}\lambda, a_0, a_{+}\lambda]$ down-tilt, no change, up-tilt the current down-tilt of magnitude λ. The minimum downtilt angle is 1° and the maximum is 14°. These are chosen to avoid excessive uptilt/downtilt [46].
- Handover: This action handovers a specific user from a specific slice from the current eNB to a nearby eNB. eNBs in LTE are interconnected with the X2 interface. If two eNBs are served by the same MME, a handover from the source to the target eNB will take place over the X2 interface. Only one UE at a time is requested to be

handed over. To maximize chances of handover success, the choice of the UE depends on the UE measurement report which contains RSRP and RSRQ values. The eNB evaluates neighboring eNBs as potential handover targets for this specific UE. The action space related to actions that optimize the traffic load for eNBs, consists of three discrete actions: $[a_{H(UE_{S1})}, a_{H(UE_{S2})}, a_{H(UE_{S3})}]$ which represent the handover of UEs from slices S_1, S_2, and S_3, respectively.

- Adjust the EPC data rate for best-effort users. To minimize the SLA violations for high priority UEs, we can reduce the aggregated maximum bit rate (AMBR) for best-effort UEs. This action is represented by $[r_{S3}]$ which is a continuous value for AMBR. We can change the rate for only one user from UE_{S3} at time *t*.

This results in a hybrid (parameterized) action space; discrete actions and continuous actions. To unify the action space, we relax the discrete actions into a continuous space using the method defined in [47]. We consider the following parameterized action space: the discrete actions are selected from a finite set $A_d = \{a_1, a_2, \ldots, a_k\}$, and each $a \in A_d$ has a set of real valued continuous parameters $X_a \subseteq \mathbb{R}$. Hence, a complete action is represented as a tuple (a, x), where $a \in A_d$ is the chosen discrete action and $x \in X_a$ is the chosen parameter to execute with action *a*. The whole action space *A* is then the union of each discrete action with all possible parameters for that action:

$$A = \bigcup_{a \in A_d} \{(a, x) | x \in X_a\} \tag{4}$$

Our DRL approach outputs a value for each of the discrete actions, concatenated with all continuous parameters, and the discrete action is chosen to be the one with the maximum output value.

Note that the previous work only considered a single category of actions, which resulted in limiting them to solving a single control problem.

3) Reward Function: The reward function is defined to guide the agent/agents to make desirable decisions in order to realize the objective of the system. Here our objective is twofold: network throughput maximization and SLA violations minimization. In response to the first objective, we include the instantaneous sum of the throughput for all *M* eNBs as the first term in Eq. (6). To achieve the second objective, we penalize the agent for any SLA violation for UEs in the system as defined below in Eq. (5) and presented as second term in the reward function.

$$p_n(t) = \begin{cases} 0, & \text{if } d_n(t) \le D_n \wedge t_n(t) \ge T_n \\ -1, & \text{otherwise} \end{cases} \tag{5}$$

where $d_n(t), t_n(t)$ represent the end-to-end delay and throughput for UE $n \in [1..N]$ respectively. D_n, T_n are the violation threshold of latency and throughput defined for each slice in Table I. For example, if the end-to-end delay for a VoIP UE is above 100*ms* at time *t*, we consider this as a violation.

To this end, we set the reward at each time step *t* as

$$log \sum_{i=1}^{M} Th_i(t) + \sigma \sum_{n=1}^{N} p_n(t) \tag{6}$$

Algorithm 1 ICRAN-C Training Based on DDPG

```
1: Randomly initialize actor network $\mu$ and critic network $Q$
   with parameters $\theta^{\mu}$ and $\theta^{Q}$ respectively.
2: Initialize target actor network $\mu_t$ and target critic network
   $Q_t$ with parameters $\theta^{\mu t} = \theta^{\mu}$ and $\theta^{Qt} = \theta^{Q}$, respectively.
3: Initialize a replay buffer R with a capacity $C$ and threshold
   $T$.
4: for $episode = 1, \ldots, K$ do
5:     Receive initial observation state $s_0$
6:     for $t = 1, \ldots, T$ do
7:         Select action $a_t = \mu(s_t) + \eta$ according to the
           current policy $\theta^{\mu}$ and exploration noise $\eta$.
8:         Execute action $a_t$ and observer reward $r_t$ and new
           state $s_{t+1}$
9:         if $SizeofR > T$ then
10:            Sample a random minibatch of $N$ transitions:
                   $N = R.sample(< s_i, a_i, r_i, s_{i+1} >)$
11:            Set $y_i = r_i + \gamma Q^{'}(s_{i+1}, \mu^{'}(s_{i+1}|\theta^{\mu^{'}})|\theta^{Q^{'}})$
12:            Update critic $Q$ by minimizing loss function:
                   $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^{Q}))^2$
13:            Update the actor $\mu$ by applying policy gradient:
   $\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{si}$
14:            Update target networks:
                   $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$
                   $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$
15:        end if
16:    end for
17: end for
```

We use both log function and σ, which is a positive weight, to balance between the two objectives.

C. ICRAN-C via Single-Agent DRL

In the single-agent DRL setting, owing to our continuous action space, we choose the widely used DDPG algorithm to find the optimal policy. DDPG is specifically adapted for problems with a continuous action space [48], unlike the DQN, which only works in environments with a discrete action space. DDPG is a model free algorithm because the agent cannot predict the future states of the environment without taking the action. Besides, it is an off-policy method because the policy used to improve the Q-function approximation is different from the behavior policy, used to explore the environment.

We list in Algorithm 1 the DDPG algorithm we use for training the agent. DDPG follows a critic-actor approach [49], in which an actor algorithm tries to output the best action and a critic tries to predict the value function for this action. The DDPG algorithm maintains a parameterized actor function μ to specify the current policy by deterministically mapping states to a specific action. The critic Q is learned using the Bellman equation. First, we initialize all of the critic Q and the actor networks μ with random values of θ^{μ} and θ^{Q} respectively. Then, we start our iterative training process (Line 4). The critic network is trained to simulate the real Q-table using neural networks. The actor network is trained to generate a deterministic policy instead of the policy gradient which chooses a random action from a determined distribution. For computing optimization, the algorithm is learning in minibatches, rather than online, therefore we initialize a replay buffer R with fixed size in Line 3. Moreover, the replay buffer is used to store the transitions that are sampled from the environment according to the exploration policy and the tuple $<s_i, a_i, r_i, s_{i+1}>$ in Line 10. When the replay buffer is full the oldest samples are discarded. Within each time step t, we select an action according to the policy μ with a certain random noise (η) and we execute it as an exploration to find the best solution(Line 7). We store the transition in the replay buffer R (Line 10) accordingly. Note that in the forward pass, we compute a loss function (Line 12) to update the critic by minimizing such loss over the chosen random transition i chosen from the replay buffer R. We also, update the policy μ using policy gradient (Line 13), through the soft-update of the policy and critic parameters θ^{Q} and θ^{μ} respectively (Line 14).

Algorithm 2 ICRAN-D Training Based on MADDPG Algorithm for M eNBs

```
1:  Initialize a replay buffer R with a capacity $C$ and threshold $T$.
2:  for $episode = 1, \ldots, K$ do
3:      Initialize a random process N for action exploration
4:      Receive initial observation state $x$
5:      for $t = 1, \ldots$,max-episode-length do
6:          for each eNB $i$ Select action $a_i = \mu(s_i) + \eta$ according to the current policy $\theta^{\mu}$ and exploration noise $\eta$.
7:          Execute actions $a = (a_1, \ldots, a_M)$ and observe reward $r$ and new state $x'$
8:          Store $(x, a, r, x')$ in replay buffer $R$
9:          $x \leftarrow x'$
10:         for eNB $i = 1, \ldots, M$ do
11:             Sample a random minibatch of $N$ transitions:
                    $N = R.sample(< x^j, a^j, r^j, x'^j >)$
12:             Set $y^j = r_i^j + \gamma Q_i^{\mu'}(x'^j, a'_1, \ldots, a'_M)|_{a'_k=\mu'_k(s_k^j)}$
13:             Update critic $Q$ by minimizing loss function:
                    $L = \frac{1}{N}\sum_j (y^j - Q_i^{\mu}(x^j, a_1^j, \ldots, a_M^j))^2$
14:             Update the actor $\mu$ by applying policy gradient:
                    $\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{si}$
15:         end for
16:         Update target networks for each eNB $i$:
                $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
17:     end for
18: end for
```

D. ICRAN-D via Multi-Agent DRL

Multi-Agent Reinforcement Learning (MARL) involves using several agents at the same time. The simplest approach in multi-agent settings is to use agents that learn and act independent of each other. We attempted this approach, by having an agent per eNB, but it did not perform well and was generally unstable. This happened because each agent's policy changes during training, resulting in a non-stationary environment. In other words, a policy change by an agent will influence the policy of the other agents and hence the lack of coordination will lead to conflicting policies. For example, handover actions produce ping pong effects and antenna tilt optimization actions produce coverage holes. Accordingly, we need to enable the agents to communicate their actions to each other. We have therefore formulated our distributed control problem as a cooperative multi-agent DRL problem, where the agents interact with each other, and their reward depends on their joint behavior. Knowing the actions taken by all agents makes the environment stationary even when policies change.

The training of multiple agents has long been a computational challenge. Since the complexity in the state and action space grows exponentially with the number of agents, even modern deep learning approaches may reach their limits [43]. If the training of agents is applied in a centralized manner, all information such as actions, observations and rewards from all gents should be sent to a centralized unit. In contrast to the centralized scheme, the training can also be handled in a distributed fashion where each agent performs local updates on and develops an individual policy without utilizing foreign information and this approach is infeasible in our work due to the non-stationarity problem. Therefore, we recognize another training scheme adopted by [50], [51]; a centralized training and a decentralized execution. This approach assumes the existence of a centralized controller that collects extra information about the agents to ease training but not used during the normal operation of the system.

To realize the multi-agent proposal, we adopt the Multi-agent DDPG framework (MADDPG) that was proposed by OpenAI in [50]. MADDPG extends DDPG into a multi-agent policy gradient algorithm where decentralized agents learn a centralized critic based on the observations and actions of all agents. Each agent has local information and local policies to train, but the centralized critic advises the agents on how to update their policies. The critic is augmented with extra information about the policies of other agents which loosen the non-stationarity of the environment. After the training is completed, the centralized critic is no longer needed; only the local actors are used in the testing phase. Here in algorithm 2, we consider a system with M agents, which represents eNBs, with policies parameterized by $\theta = \{\theta_1, \ldots, \theta_M\}$, and let $\pi = \{\pi_1, \ldots, \pi_M\}$ be the set of agents' policies. $Q_i^{\pi}(x, a_1, \ldots, a_M)$ is a centralized Q-function that takes as input the actions of all agents, $a_1, \ldots, a_M$, in addition to some state information x, and outputs the Q-value for agent i. In our problem, x consists of the state space of either all eNBs or only the neighbouring eNBs (agents) to reduce the communication overhead. The first lines in the algorithms (1-7) are for parameters initialization and exploration. MADDPG uses a replay buffer to store the agent transitions (x, a, r, x') (Line 8). Then, a batch of these transitions is sampled from the experience replay to train agent i (Line 11). Line 13 is used to update an agent's centralized critic by minimizing the loss function. Note that the centralized critic uses joint information to update its parameters. Similar to DDPG in algorithm 1, MADDPG uses the deterministic policy gradient to update each of the agent's i actor parameters (Line 14). We take the gradient with respect to the actor's parameters using a centralized critic as guidance as shown below:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{s \sim p^{\mu}, a_i \sim \pi_i}[\nabla_{\theta_i} log\pi_i(a_i|s_i) Q_i^{\pi}(x, a_1, \ldots, a_M)] \tag{7}$$

The most important thing to notice is that even though the actor only has local observations and actions, the use of a centralized critic during the training phase provides information about the optimality of the agent's actions for the entire system.

E. Complexity Analysis

The complexity and implementation overhead should be kept low. This overhead may arise due to excessive signaling associated with exchange of data between ICRAN and the environment. In ICRAN-C, we assume that the network state is fully observable by the agent. During every step execution ICRAN-C collects the current state of all M eNBs in the network which results in communication overhead of $\mathcal{O}(M)$. In contrast, in ICRAN-D which is based on multi-agent DRL, we assume that the agents are communicating with each other to reach the final goal simultaneously. We propose two ways of coordination between the multiple agents; 1) exchanging information with all eNBs in the system; thus the signaling overhead for each agent is $\mathcal{O}(M-1)$. 2) exchanging information with nearby $\acute{M}$ eNBs ($\acute{M} << M$), here the signaling overhead for each agent is $\mathcal{O}(\acute{M}-1)$. In addition to the eNB information such as antenna tilt and transmitted power, the exchanging information also contains a list of UEs' performance (i.e., throughput and delay) and their slicing profile. Hence, the size of the message between the eNB and the DRL agent in both ICRAN-C and ICRAN-D is $\mathcal{O}(P)$ where P represents the number of UEs associated with an eNB. This corresponds to a few megabytes of data for an eNB serving a million UEs.

VI. Experimental Setup

For our experiments, we used the well known ns-3 network simulator. For the reinforcement learning we used OpenAI Gym, which is a toolkit for developing RL algorithms. To integrate our network environment with Gym, we used ns3-Gym framework [52] which simplifies exchanging observations, actions and rewards between the RL agent and the network environment (See Figure 3). We implement two different architectures for ICRAN as shown in Figure 4.

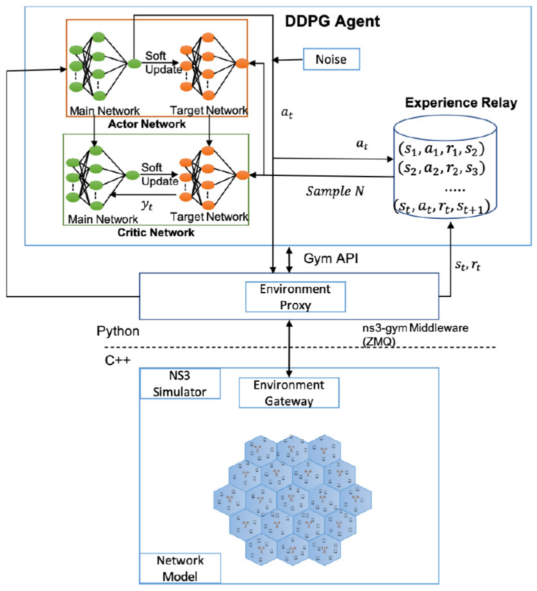


Fig. 3. Experimental Setup.

A. Network Environment

We evaluated our proposed framework in an environment consisting of 19 eNBs shown in Figure 4. We divided users evenly amongst the three slices in use: VoIP, video and best-effort. Further, we simulated VoIP using an ON/OFF model; ON is the time when users are speaking and OFF is the time when the users are not. We set the On-Time and Off-Time to 0.352 seconds and 0.650 seconds, respectively, to ensure a bit rate of 64 Kbps. The video traffic was simulated using Evalvid module[1] which streams video frames. Finally, we setup the best-effort application using the UDP echo module in ns-3. The VoIP traffic was mapped to QCI 1, the video traffic was mapped to QCI 7 and the best-effort traffic was mapped to QCI 9 and each user received traffic from only one application. Users exchange traffic with end points outside the mobile networks (i.e., the remote hosts in Figure 1).

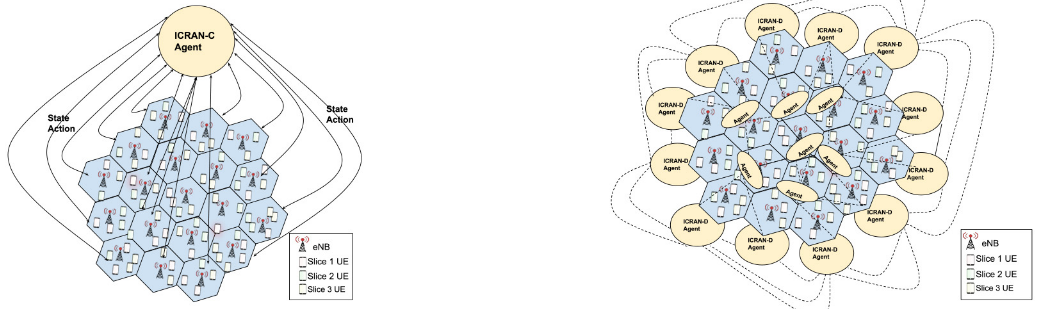
The arrival of users of each slice S followed a Poisson process with an arrival rate of $\lambda = 5$ *users/S/eNB* throughout the coverage area of the cell. 50% of the users were stationary, 25% were vehicular users moving with an average speed of 30 km/h and the rest 25% were walking at 0.8 *m/s*. Table II summarizes the simulation parameters.

B. Simulation Scenarios

For our evaluation, we considered two specific scenarios that capture extreme network conditions to help assessing the effectiveness of the proposed control mechanism.

1) Network Congestion: When the required resources by all the slices are less than the resources provided by the

[1]https://gitlab.com/gercom/evalvid-ns3



(a) ICRAN-C (b) ICRAN-D

Fig. 4. High-Level architecture for ICRAN including the network topology: a) ICRAN-C: a centralized single-agent receives a state from each eNB and chooses an action to perform to get the reward. b) ICRAN-D: partially decentralized multi agents controllers; each agent is associated with an eNB with some communication between the agents.

TABLE II
SIMULATION PARAMETERS

Parameter	Value
Number of eNBs	19
Cell Radius	250 m
eNBs Distance	500 m
Antenna type	2x2 MIMO
System Bandwidth	5MHz
eNB Tx power	30 dBm
UE Tx power	10 dBm
Mobility Model	Random Walk
Vehicular UE speed	30 km/h
Pedestrian UE speed	0.8 m/s
MAC Scheduler	Proportional Fair
Simulation Duration	500 Seconds

network, the decision making by the controller is easy. Therefore, we choose high network loads scenarios that require from the controller to obtain a complete image of the network and make intelligent decisions to achieve high end-to-end performance while minimizing SLA violations.

2) Network Failure: In this scenario, we generate some cell faults during the simulation to evaluate whether the proposed control mechanism can adapt quickly to the changes. Specifically, we randomly assign faults like excessive uptilt/downtilt to one cell. Excessive uptilt/downtilt is simulated using extreme values for the antenna downtilt which are $[0, 1]^\circ$ for excessive antenna uptilt and $[16, 15, 14]^\circ$ for excessive antenna downtilt.

C. Reinforcement Learning Setup

In this experiment, inspired by the work in [48], we used fully connected neural networks to implement the actor and critic of DDPG and MADDPG that we describe in Algorithms 1 and 2. For the actor network, we use one input layer, two hidden layers and one output layer. The neuron numbers for these layers are 16, 64, 32, 8, respectively. The critic network comprises a one input layer, one hidden layer and one output layer with neuron numbers of 24, 64, 1, respectively. For the single agent training, we set the learning rate of actor and critic in Algorithm 1 to 0.0001 and 0.001, respectively. While for multi-agent training, we use learning rate of 0.0005 and 0.001 for actor and critic networks in Algorithm 2. Additionally, for both the DDPG and MADDPG algorithms, we sample after every other 100 timesteps, and sample a batch size of 32 by episode using replay buffer size of 10000. We set τ, soft update of target, to 0.01, and the discount factor γ to 0.99 , which places more focus on the immediate reward. To balance the two components of the reward function we use $\sigma = 1.25$. In order to find suitable values for the hyper-parameters, we start with the original values proposed in [48], [50]. Then through extensive simulations and grid search, we adjusted some of the values based on the performance of the algorithms. All these parameters are summarized in Table III.

TABLE III
HYPER-PARAMETERS FOR ICRAN

Parameter	Value
Discount factor (γ)	0.99
Critic learning rate	0.001
Actor learning rate	0.0001
Replay buffer size (C)	10000
Batch size (N)	32
Soft update of target (τ)	0.01

VII. PERFORMANCE EVALUATION

A. Overview of the Evaluation

ICRAN is evaluated based on whether it achieves a network performance that satisfies the SLA requirements for slices while maximizing radio resource utilization. We conduct four stages of evaluation. The first stage looks at the training phase of ICRAN and its convergence performance (Section VII-B). The second stage investigates the decisions taken by the ICRAN agent to minimize the number of SLA violations after training and why ICRAN outperforms other decision-making approaches (Section VII-C). In the third stage, we compare ICRAN to some baselines and state-of-the-art methods in terms of physical resources utilization and SLA satisfaction (Section VII-D and Section VII-E). Finally, we study the generalized performance of ICRAN in terms of throughput and

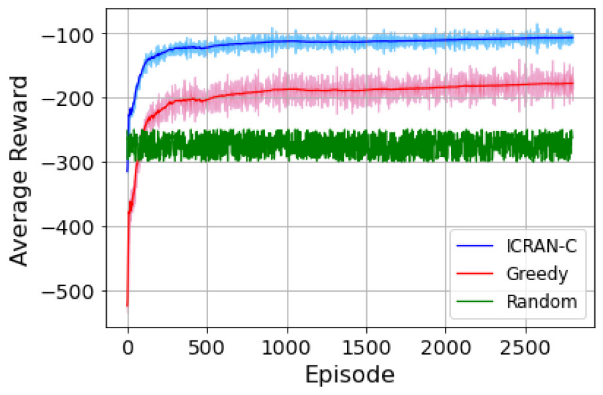


Fig. 5. The average accumulated reward that is achieved by ICRAN-C, Greedy and Random approaches. The envelope shows standard deviation.

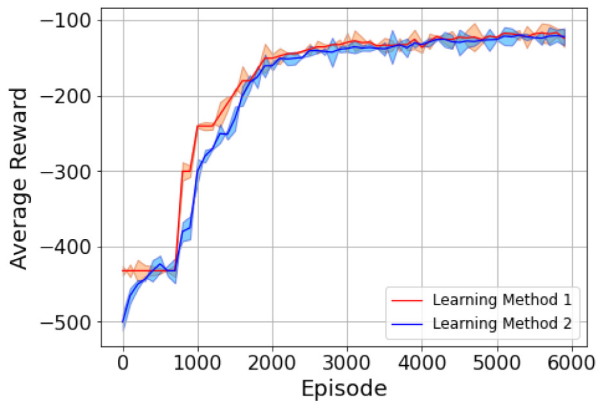


Fig. 6. The average reward of ICRAN-D for two different learning methods in the training. Method 1: Critic network of any agent receives all policy information of all agents in the system. Method 2: Critic network of any agent only receives policy information of nearby agents in the system. The envelope shows the standard deviation.

delay under different levels of network load (Section VII-E). Every scenario in this section is simulated 50 times.

B. Training Performance

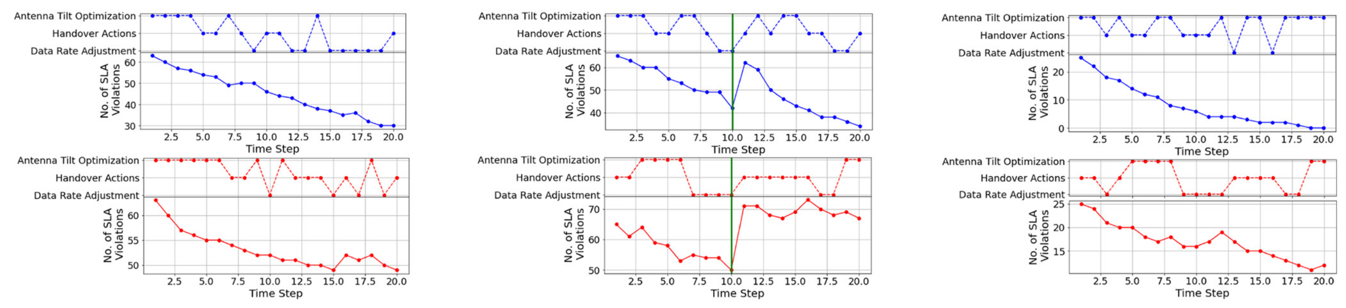
We compare the average reward and convergence performance of the ICRAN-C to those of a random and a greedy methods. The random method selects actions in the action space in a uniformly random fashion. The greedy algorithm picks the best action without taking into consideration the long-term effect of that decision. It essentially measures the immediate impact of taking an action on throughput and SLA violations. Then updates a state action table to track how effective an action is. To add some randomness, which allows the algorithm to explore the action space, we use an epsilon-greedy algorithm which adds randomness when deciding between actions. The algorithm chooses the action randomly with a probability ϵ which will make the algorithm explores other actions as well. Most of the time, the algorithm selects the best action which result in a high reward with probability $1 - \epsilon$. We try different values for ϵ and choose 0.1, which results in the best reward. Figure 5 shows that the ICRAN-C achieves a significantly higher reward than the other two alternatives. Reaching this reward level happens relatively fast, after 500 training episodes. We also note, unlike for the other two strategies, that the achieved reward exhibits low variability.

In Figure 6, we plot the average reward for the ICRAN-D. Here, we compare two learning methods. In the first method, the centralized critic uses information from all agents (red curve), while in the second learning method it uses only information from adjacent agents (blue curve). We observe a negligible difference between the two training approaches in our simulation scenario which has 19 eNB. However, we believe a larger network topology can capture the difference between the two learning methods. There are two differences between ICRAN-D and ICRAN-C. The former takes $4x$ the time the latter takes to converge. This difference holds, even if we change the learning rate. Further, the average reward of ICRAN-D is slightly lower but clearly higher than the greedy and random strategies.

C. Why Does ICRAN Outperform Other Methods?

ICRAN clearly achieves a higher reward compared to the greedy approach, which resembles the way a human operator would behave. Thus, it is important to understand the strategies that ICRAN follows to achieve this. To this end, we explore the way ICRAN and the greedy approaches behave under three different scenarios. A scenario is a single simulation snapshot, that is one realization, that lasts 20 time steps. Considering a single realization helps in gauging the impact of each action. The three scenarios are an overloaded network, a failure of an eNB in an overloaded network and a network that is loaded slightly below its full capacity. For each scenario, we track the actions that different approaches take and their impact on reducing SLA violations.

For Snapshot 1 (see Figure 7a), which represents a network congestion scenario with a network load level of 120%. ICRAN-C (blue curve) is able to reduce the SLA violations from 63 to 30, while the greedy algorithm (red curve) ends up with 49 violations. As expected, there is a mismatch between the actions taken by the two approaches. We observe that, unlike the greedy approach, some of the actions that ICRAN-C takes do not result in an immediate reduction in SLA violations. However, these actions lead to a significant drop in SLA violations in the following time steps. An example of this is the consecutive antenna tilt optimization, handover and data rate adjustment actions that are executed at time steps 7, 8 and 9. In the second snapshot in Figure 7b, we simulate the same network load level, i.e., 120% and at time step = 10 we introduce a failure in one eNB. The number of SLA violations jumps following the failure. The greedy algorithm handles this situation by following a sequence of handover actions to move the UEs attached to the failed eNB to nearby eNBs. This strategy minimizes the SLA violations by one at each time step. However, the ICRAN-C alternates between antenna tilt optimization and handover actions. We can observe that data rate adjustments actions are unlikely taken by ICRAN-C in this situation. ICRAN-C's strategy is to increase the coverage of the nearby eNBs' by adjusting their antenna tilt and at the



(a) Snapshot1: Network load level=120%
(b) Snapshot2: Network load level=120% with failure in one eNB at time step=10
(c) Snapshot3: Network load level=90%

Fig. 7. Action strategies taken by ICRAN-C(blue) and the greedy algorithm (red) under three testing scenarios.

same time rely on the automatic handover that the LTE system provides. This strategy helps ICRAN-C to mitigate the failure in a few time steps. Finally, the third scenario, which we depict in Figure 7c, corresponds to a network that is loaded at 90%. Accordingly, an optimal control algorithm must be able to ensure zero SLA violations. ICRAN-C indeed manages to achieve that, while the greedy algorithm fails. The main reason is that the greedy algorithm chooses to take the same action sequentially through time steps as long as the SLA violations decrease. This strategy leads to a slow decrease or increase in the number of SLA violations.

Takeaways: ICRAN outperforms the greedy approach in minimizing SLA violations. It achieves this by executing control actions that do not only focus on the immediate reduction of SLA violations. Another aspect to ICRAN is its alternation between available control actions. This strengthens the case for multi-action control strategies that prioritize long term reward. Note that the greedy strategy resembles the manual troubleshooting that is common in today's networks.

D. Resource Utilization Efficiency

Having seen that ICRAN converges to finding optimal control sequences, we now turn to evaluating ICRAN in practice. We check whether ICRAN can lead to an efficient resource utilization in comparison to two default approaches that resort to resource reservation and over provisioning. The first approach, which we call *Static Slicing* allocates 50%, 40% and 10% of the RBs to S_1, S_2 and S_3, respectively. The traditional LTE scheduler is then used to assign RBs within each slice. The second approach, *Dynamic Slicing*, besides assigns RBs to slices, it adaptively reassign RBs that are not used by a higher priority slice to a lower priority one. Note that ICRAN combines dynamic slicing and DRL (see Section V). Moreover, we compare ICRAN against two recent works from state-of-the-art. We choose these methods because they employ diverse approaches and have shown excellent performance in RAN resources management among network slices. The first work is DeepSlicing [25] that leverages a DRL algorithm, namely DDPG, to allocate the radio resources to users within a slice. For each slice there is a DRL agent that learns the optimal policy of allocating resources to users by observing the users' utility. The agent is penalized if the minimum utility requirement of users is not satisfied. Coordination of physical resources among network slices is formulated as a quadratic optimization problem aiming to maximize the sum-utility of all network slices. We implement and simulate DeepSlicing in our ns3 environment setup in Section VI; we implement three DDPG agents; each for each slice to allocate the resources to users in the network slice, i.e., action space. The reward function is penalizing the DDPG agent if the minimum utility requirement of the users is not satisfied. On the top of the DDPG agents, we solve the slices coordinating problem as a quadratic programming using the optimization tool CVXPY[2] which is an open source Python-embedded library for convex optimization problems. We train the three DDPG agents on one eNB until convergence, and then use them in all eNBs during inference. The second work is TNSM-21 [24] which is a data-driven resource management method to support RAN slicing. Based on monitoring RAN information namely, CQI reports collected from the base stations, they calculate the amount of physical resources to allocate per slice to meet the target KPIs. Similar to our proposed slicing model in Section IV, in this work the authors presented a slice orchestrator that is responsible for managing slices. Based on the UE channel quality from the CQI report, the slice orchestrator translates the CQI to the maximum data rate per RB based on [53]. Two different algorithms are being proposed for the calculation of the number of RBs per slice based on the slice requirements. For guaranteed latency slice, the number of RBs are calculated based on the queue model M/M/1/K to estimate the latency of the packets. Beside, the average packet size of the latency-constrained application and the maximum data rate provided by one RB, we can calculate the number of RB for this slice. For throughput guaranteed slice, the number of RBs is equal to (or greater than) the aggregate data rate needed by the slice for all users. We implement this algorithm in our slice orchestrator without implementing CQI overhead optimization method defined in this work. We frequently monitor the CQI value for users and map it to the maximum data rate for RB. However, we add additional case to the algorithm to cover the third best-effort slice, which assigns the remaining resource blocks to this slice.

Figure 8 shows the fraction of utilized RBs by each approach under different levels of network load. Both static and dynamic slicing achieve lower RBs utilization compared

[2]https://www.cvxpy.org/

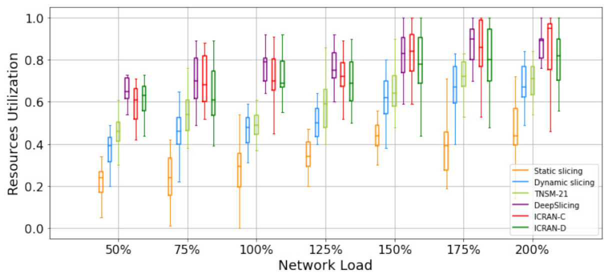


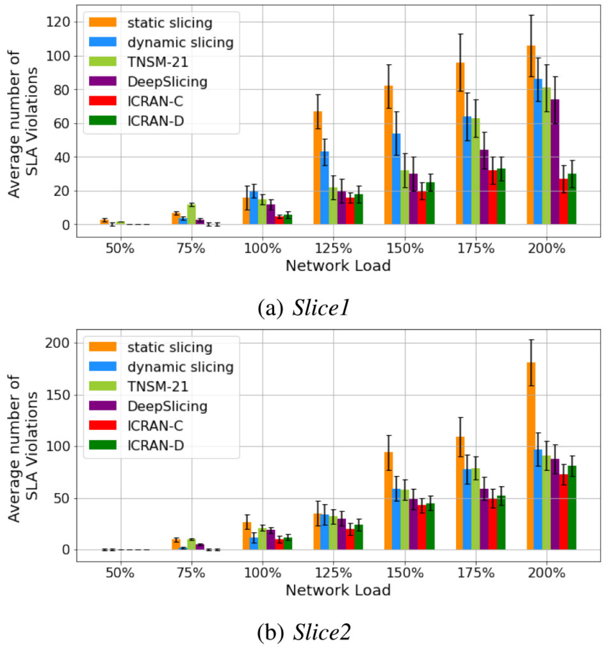
Fig. 8. Resource blocks utilization under different levels of network load.

with other methods. As expected dynamic slicing outperforms static slicing, which fails to fully utilize available resources even at lower levels of load due to its rigidity. TNSM-21 shows improvement in RBs utilization efficiency over dynamic slicing because it allocates only the required number of RBs that satisfy the slice requirements. However, because the RBs are reserved, they are not actually used until needed, thus there is room for improvement. DeepSlicing outperforms all methods at low network loads, i.e., 50%, 75% and 100% because it optimizes the RBs between the slices besides the resources allocation to individual users within the slice. The difference between DeepSlicing and ICRAN is marginal though. In a highly congested network, i.e., 200% load, ICRAN-C results in 97% median RBs utilization efficiency, which is higher than DeepSlicing's. This is because ICRAN-C attempts to perform a network-wide optimization rather than local optimization. DeepSlicing outperforms ICRAN-D, as the latter lacks the overall view of the network compared to ICRAN-C.

Takeaways: The DRL-based methods clearly outperforms the other non-DRL method. DeepSlicing and ICRAN-C demonstrate a comparable efficiency with DeepSlicing leading at lower network load (i.e., 100% or less) values and ICRAN-C leading at extreme levels of load, which is more challenging.

E. Minimization of SLA Violations

We took a closer look at the performance of ICRAN in minimizing the number of SLA violations. The two panels in Figure 9 show the number of SLA violations for the top two slices in terms of priority (*Slice1, Slice2*) achieved by different methods for different levels of network congestion. The basic slicing methods, i.e., static and dynamic demonstrate a poor performance in minimizing the number of SLA violations because they do not consider satisfaction of the slice requirements, when allocating RBs to a slice. However, we observe that DRL-based methods, namely ICRAN-C, ICRAN-D and DeepSlicing outperform TNSM-21 for both slices because this approach does not leverage any domain knowledge about the nature of the available actions, the system's state transition dynamics, and its reward function. Therefore, TNSM-21 can only allocate the minimum resources calculated by the algorithm and serve accordingly a specific number of users in *Slice1* and *Slice2*. Overall, ICRAN-C and ICRAN-D consistently outperform DeepSlicing. This difference becomes more remarkable as the network load increases. For example, when the network is 200% loaded, DeepSlicing results in 64% and 59% higher SLA violations in *Slice1* compared to ICRAN-C and ICRAN-D, respectively. The is because of the large action space for ICRAN. While ICRAN is giving priority to *Slice1*, DeepSlicing is treating both slices equally based on their defined SLA. However, ICRAN-C outperforms ICRAN-D in reducing the number of SLA violations, this is mainly due to the design of ICRAN-C which has a full observability of the network, while ICRAN-D has a partial observability where agents have access only to information of nearby eNBs. While the difference is generally small, both approaches respect the priority of slices. We note the trade-off between achieving efficient resource utilization and the minimization of SLA violations. ICRAN succeeds in achieving both, while DeepSlicing focuses on the former, hence its slightly better performance in RB utilization when the network is lightly to moderately loaded. However, this better performance does not translate to the best performance in terms of reducing SLA violations.



(a) *Slice1*

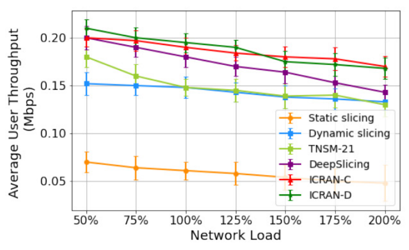
(b) *Slice2*

Fig. 9. Number of SLA violations, when using ICRAN-C, ICRAN-D, DeepSlicing and TNSM-21, as network load varies for (a) *Slice1* and (b) *Slice2*.

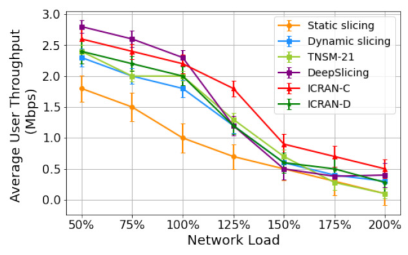
Takeaways: ICRAN outperforms the other methods in minimizing the number of SLA violations in *Slice1* and *Slice2* and considers the priority of the slices in allocating resources. The remarkable SLA violations reduction of ICRAN can be attributed to, (i) ICRAN incorporates the violation of slices' SLA into the reward function, (ii) ICRAN optimizes for the overall network which allows to utilize the physical resources efficiently to minimize the number of SLA violations for the whole network, and (iii) the larger action space that allows for quickly finding a sequence of actions that strike a balance between quality assurance and resource utilization. This confirms the need for a multi-objective reward function and a larger action space.

F. Network Performance

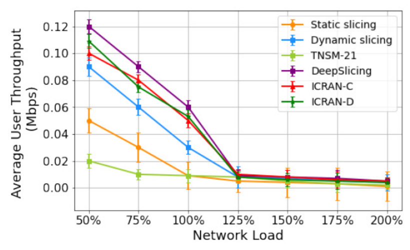
We now proceed to quantify the throughput and delay experienced by users in different slices, when using ICRAN, static



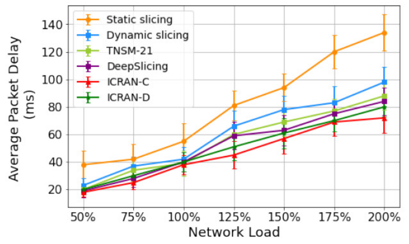
(a) Average user throughput Slice1 (VoIP traffic)



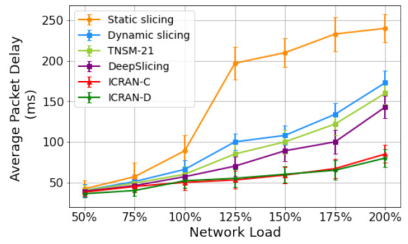
(b) Average user throughput Slice2 (Video traffic)



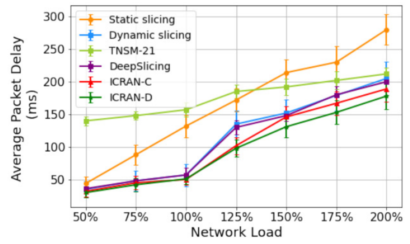
(c) Average user throughput Slice3 (BE traffic)



(d) Average packet delay Slice1 (VoIP traffic)



(e) Average packet delay Slice2 (Video traffic)



(f) Average packet delay Slice3 (BE traffic)

Fig. 10. Network performance under different levels of network load.

slicing, dynamic slicing, DeepSlicing and TNSM-21 as the network load increases. Figure 10 shows the average user throughput and average packet delay per slice for different levels of network load along with the standard deviation. All methods try to maintain a constant throughput for the VOIP slice. However, ICRAN ensures both the highest throughput of $0.143 Mbps (\pm 0.01)$ and lowest delay of $72 ms (\pm 9)$ when the network load is 200% (see Figures 10a and 10d).

In the case of video traffic, as the network load increases, the average user throughput for all methods falls (see Figure 10b). For high network loads, ICRAN-C outperforms all approaches, while DeepSlicing achieves a slightly higher throughput than ICRAN-C when the number of users per slice is low. For example, when the network load is 200%, ICRAN-C outperforms DeepSlicing by 20% in throughput. In contrast, when the network load is 50%, Deepslicing results in 7% higher throughput than ICRAN-C and 15% than ICRAN-D. This is because DeepSlicing's agent is awarded based on the maximum throughput per base station. The agent observes only the corresponding base station and cannot recognize the state of other base stations to take further actions when the network is congested. TNSM-21 exhibits the same results as dynamic slicing with little improvement. When looking at delay both ICRAN approaches clearly outperform the other four methods (see Figure 10e). When the network load is 200%, ICRAN results in an average delay that is 200%, 116%, 100% and 79% lower than that of static slicing, dynamic slicing, TNSM-21 and DeepSlicing, respectively. This is due to the reward function defined in ICRAN which directly optimizes for both throughput and delay.

Finally, for the best effort traffic, the average throughput decreases linearly, for all methods, as the network load increases from 50% to 125% (Figure 10c). ICRAN achieves a higher throughput for load levels lower than 125%. The throughput drops to a minimal level for load levels higher than 125%. For ICRAN, this is expected, since it reduces the data rate for best effort UEs to minimize the SLA violations for high priority slices. TNSM-21 achieves lower throughput regardless of the network load level, because the method is designed to allocate RBs to throughput-constrained and delay-constrained slices. The remaining RBs are allocated to best-effort UEs. There are no differences in delay between ICRAN and the other methods when the network is not loaded, we however record a slight difference, in favor of ICRAN, as the load increases (see Figure 10f). We also note that the benefit from using ICRAN is not limited to high load cases. It consistently delivers a better performance at all levels of load.

Takeaways: ICRAN demonstrates its superiority in network performance in terms of throughput and delay for the different slices. For VOIP traffic, all methods try to maintain a constant throughput, however, both ICRAN approaches achieve approximately 71%, 22%, 24% and 16% higher throughput compared with static slicing, dynamic slicing, TNNSM-21 and DeepSlicing when the network is overloaded. Also, ICRAN ensures the minimum delay in VOIP slice. In video slice, DeepSlicing shows a slight improvement in throughput over ICRAN when the number of users is low. This is due to DeepSlicing reward function that maximize for throughput at the base station. However, when the number of user increases, ICRAN performs better. For the delay, ICRAN results in the minimum delay under different network loads. ICRAN exhibits the same behavior for best-effort traffic.

VIII. Discussion

Self-driving RAN: We have demonstrated that it is possible to develop a multi-objective automated control mechanism for managing the performance in a multi-slice RAN. In the course of this work, we have identified a number of key insights that we believe are pertinent to efforts that aim to realize a fully self-driving RAN. First, the key insight behind ICRAN is that the success in achieving the control objectives is due to the collective behaviour of all eNBs in the network. This is evident when comparing ICRAN to slicing methods and other recent works. We accordingly believe that control strategies that depend on the joint decision instead of the independent decisions of system components will be key to realizing self-driving RANs. Second, ensuring an optimal or near optimal network-wide control requires strategies that look beyond the immediate reward. Considering several time steps ahead helps avoiding actions that only result in local optimization but lead to service degradation in other parts of the network. Accordingly, heuristics and threshold-based control approaches will always likely fail in ensuring such optimality. Third, the success of ICRAN can also be attributed to the design of the reward function and the use of three different types of actions, which are both novel. Our reward function is able to guide the agent to the optimal decision through exploration and exploitation of various actions from different categories. The key takeaway point here is that a multi-objective control requires a diverse set of actions and reward functions that reflect that.

Implementation Consideration: Although, simulations confirm the effectiveness of ICRAN, it must be also practically implementable. In general, DRL does not make strong assumptions about the target system, however, we have some simulation-based assumptions, which we need to address in the real implementation. For example, we need to define the communication channels between the eNBs and the centralized agent in case of ICRAN-C and between the agents in ICRAN-D. Such channels and the respective protocols can be implemented as application level services to avoid the need for adding them to the standards. We believe that the O-RAN architecture, due to its flexibility, can ease the task of implementing such protocols [54]. Furthermore, ICRAN assumes the availability of detailed telemetry to track the state of the network. Such telemetry does not exist today but new approaches to telemetry like in-band telemetry seem promising since they balance overhead and utility [55].

Limitations and the way ahead: Even though ICRAN has succeeded in achieving its aim to optimize the overall network performance while satisfying the QoS requirements in multi-slice RAN, we highlight a number of limitations and enhancements that we plan to address in the future work. First, to support the increasing heterogeneous services and complex networks, we need to include other types of traffic with different patterns such as IoT. Second, we need to investigate the impact of the control interval on ICRAN decisions. We examine values of 1 *sec*, 5 *secs* and 10 *secs*, and decide to use 10 *secs* since this matches the time needed for performing our control actions. Shorter timings yielded poor performance results. Another issue is that our topology is relatively small, which may raise concerns about whether ICRAN can scale to much bigger networks. We believe that our preliminary results which have shown minimal differences between fully and partial observability distributed learning approaches are promising.

IX. Conclusion

In this paper, we have presented ICRAN, a novel control framework for optimizing resources utilization while minimizing SLA violations in a multi-slice RAN. Inspired by the remarkable achievements of deep reinforcement learning in solving complex control problems in highly dynamic environments such as mobile network, ICRAN comprises two DRL-based architectures: centralized ICRAN and distributed ICRAN. Through extensive simulations using ns-3, we have confirmed the substantial advantages granted by ICRAN over other slicing schemes and recent works in terms of resources utilization and QoS assurance. ICRAN is, to the best of our knowledge, the only framework that simultaneously addresses multiple RAN problems.

References

[1] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5G: Survey and challenges," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 94–100, May 2017.

[2] "Network Automation: Efficiency, Resilience, and the Pathway to 5G." MIT. [Online]. Available: https://www.technologyreview.com/s/613533/network-automation-efficiency-resilience-and-the-pathway-to-5g/ (Accessed: Nov. 10, 2021).

[3] A. G. Spilling, A. R. Nix, M. A. Beach, and T. J. Harrold, "Self-organisation in future mobile communications," *Electron. Commun. Eng. J.*, vol. 12, no. 3, pp. 133–147, 2000.

[4] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," 2017, *arXiv:1710.11583*.

[5] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia, "Intelligence and learning in O-RAN for data-driven NextG cellular networks," *IEEE Commun. Mag.*, vol. 59, no. 10, pp. 21–27, Oct. 2021.

[6] F. Vannella, G. Iakovidis, E. Al Hakim, E. Aumayr, and S. Feghhi, "Remote electrical tilt optimization via safe reinforcement learning," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2021, pp. 1–7.

[7] Y. S. Nasir and D. Guo, "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2239–2250, Oct. 2019.

[8] J. Mei, X. Wang, K. Zheng, G. Boudreau, A. B. Sediq, and H. Abou-Zeid, "Intelligent radio access network slicing for service provisioning in 6G: A hierarchical deep reinforcement learning approach," *IEEE Trans. Commun.*, vol. 69, no. 9, pp. 6063–6078, Sep. 2021.

[9] Y. Kim and H. Lim, "Multi-agent reinforcement learning-based resource management for end-to-end network slicing," *IEEE Access*, vol. 9, pp. 56178–56190, 2021.

[10] A. P. Iyer, L. E. Li, and I. Stoica, "Automating diagnosis of cellular radio access network problems," in *Proc. 23rd Annu. Int. Conf. Mobile Comput. Netw.*, 2017, pp. 79–87.

[11] G. Brockman *et al.*, "OpenAI gym," 2016, *arXiv:1606.01540*.

[12] "ns-3 Network Simulator." [Online]. Available: https://www.nsnam.org/ (Accessed: Nov. 10, 2021).

[13] M. Iwamura, "NGMN view on 5G architecture," in *Proc. IEEE 81st Veh. Technol. Conf. (VTC Spring)*, 2015, pp. 1–5.

[14] M. Richart, J. Baliosian, J. Serrat, and J.-L. Gorricho, "Resource slicing in virtual wireless networks: A survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 462–476, Sep. 2016.

[15] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 3rd Quart., 2018.

[16] S. Wijethilaka and M. Liyanage, "Survey on network slicing for Internet of Things realization in 5G networks," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 957–994, 2nd Quart., 2021.

[17] Y. Wu, H.-N. Dai, H. Wang, Z. Xiong, and S. Guo, "A survey of intelligent network slicing management for industrial IoT: Integrated approaches for smart transportation, smart energy, and smart factory," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 2, pp. 1175–1211, 2nd Quart., 2022.

[18] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[19] ETSI NFV, "Network functions virtualization: An introduction, benefits, enablers, challenges & call for action," Darmstadt, Germany, SDN OpenFlow World Congr., White Paper, 2012.

[20] T. Taleb, A. Ksentini, and B. Sericola, "On service resilience in cloud-native 5G mobile systems," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 483–496, Mar. 2016.

[21] "O-RAN: Towards an open and smart RAN," Alfter, Germany, O-RAN Alliance, White Paper, Oct. 2018.

[22] D. Nojima *et al.*, "Resource isolation in RAN part while utilizing ordinary scheduling algorithm for network slicing," in *Proc. IEEE 87th Veh. Technol. Conf. (VTC Spring)*, 2018, pp. 1–5.

[23] R. Shrivastava, K. Samdanis, and A. Bakry, "On policy based RAN slicing for emerging 5G TDD networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2018, pp. 1–6.

[24] S. Bakri, P. A. Frangoudis, A. Ksentini, and M. Bouaziz, "Data-driven RAN slicing mechanisms for 5G and beyond," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4654–4668, Dec. 2021.

[25] Q. Liu, T. Han, N. Zhang, and Y. Wang, "DeepSlicing: Deep reinforcement learning assisted resource allocation for network slicing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2020, pp. 1–6.

[26] A. Abouaomar, A. Taik, A. Filali, and S. Cherkaoui, "Federated learning for RAN slicing in beyond 5G networks," 2022, *arXiv:2206.11328*.

[27] N. C. Luong *et al.*, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.

[28] A. Feriani and E. Hossain, "Single and multi-agent deep reinforcement learning for AI-enabled wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1226–1252, 2nd Quart., 2021.

[29] L. Lei, Y. Tan, K. Zheng, S. Liu, K. Zhang, and X. Shen, "Deep reinforcement learning for autonomous Internet of Things: Model, applications and challenges," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1722–1760, 3rd Quart., 2020.

[30] C. She *et al.*, "A tutorial on ultrareliable and low-latency communications in 6G: Integrating domain knowledge into deep learning," *Proc. IEEE*, vol. 109, no. 3, pp. 204–246, Mar. 2021.

[31] F. Tang, Y. Kawamoto, N. Kato, and J. Liu, "Future intelligent and secure vehicular network toward 6G: Machine-learning approaches," *Proc. IEEE*, vol. 108, no. 2, pp. 292–307, Feb. 2020.

[32] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Netw.*, vol. 32, no. 6, pp. 50–57, Nov./Dec. 2018.

[33] A. M. Seid, G. O. Boateng, B. Mareri, G. Sun, and W. Jiang, "Multi-agent DRL for task offloading and resource allocation in multi-UAV enabled IoT edge network," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4531–4547, Dec. 2021.

[34] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2018, pp. 1–6.

[35] Y. Ren, A. Guo, C. Song, and Y. Xing, "Dynamic resource allocation scheme and deep deterministic policy gradient-based mobile edge computing slices system," *IEEE Access*, vol. 9, pp. 86062–86073, 2021.

[36] A. M. Seid, G. O. Boateng, S. Anokye, T. Kwantwi, G. Sun, and G. Liu, "Collaborative computation offloading and resource allocation in multi-UAV-assisted IoT networks: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 12203–12218, Aug. 2021.

[37] J. Pérez-Romero, O. Sallent, R. Ferrús, and R. Agustí, "Knowledge-based 5G radio access network planning and optimization," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, 2016, pp. 359–365.

[38] "5G-Lena Module." [Online]. Available: https://5g-lena.cttc.es (Accessed Nov. 15, 2021).

[39] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang, "Large-scale measurement and characterization of cellular machine-to-machine traffic," *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 1960–1973, Dec. 2013.

[40] A. Aghmadi, I. Bouksim, A. Kobbane, and T. Taleb, "A MTC traffic generation and QCI priority-first scheduling algorithm over LTE," in *Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM)*, 2015, pp. 1–6.

[41] P. H. A. Rezende and E. R. M. Madeira, "An adaptive network slicing for LTE radio access networks," in *Proc. Wireless Days (WD)*, 2018, pp. 68–73.

[42] H.-S. Chuang, S.-L. Hsieh, and C.-F. Wu, "A channel-aware downlink scheduling scheme for real-time services in long-term evolution systems," in *Engineering Innovation and Design*. Boca Raton, FL, USA: CRC Press, 2019, pp. 337–343.

[43] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: A survey," *Artif. Intell. Rev.*, vol. 55, pp. 895–943, Apr. 2021.

[44] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda, "Downlink packet scheduling in LTE cellular networks: Key design issues and a survey," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 678–700, 2nd Quart., 2012.

[45] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine Learning Proceedings*. Amsterdam, The Netherlands: Elsevier, 1994, pp. 157–163.

[46] A. Gómez-Andrades, P. Muñoz, E. J. Khatib, I. de-la Bandera, I. Serrano, and R. Barco, "Methodology for the design and evaluation of self-healing LTE networks," *IEEE Trans. Veh. Technol.*, vol. 65, no. 8, pp. 6468–6486, Aug. 2016.

[47] M. Hausknecht, P. Mupparaju, S. Subramanian, S. Kalyanakrishnan, and P. Stone, "Half field offense: An environment for multiagent learning and ad hoc teamwork," in *Proc. AAMAS Adaptive Learn. Agents (ALA) Workshop*, 2016, pp. 1–7.

[48] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[49] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2000, pp. 1057–1063.

[50] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," 2017, *arXiv:1706.02275*.

[51] J. N. Foerster, Y. M. Assael, N. De Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," 2016, *arXiv:1605.06676*.

[52] P. Gawłowicz and A. Zubow, "ns-3 meets OpenAI gym: The playground for machine learning in networking research," in *Proc. 22nd Int. ACM Conf. Model. Anal. Simul. Wireless Mobile Syst.*, 2019, pp. 113–120.

[53] *LET; Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Layer Procedures, V.15.2.0, Release 15*, 3GPP standard TS 36.213, Oct. 2018.

[54] A. Garcia-Saavedra and X. Costa-Pérez, "O-RAN: Disrupting the virtualized RAN ecosystem," *IEEE Commun. Stand. Mag.*, vol. 5, no. 4, pp. 96–103, Dec. 2021.

[55] L. Tan *et al.*, "In-band network telemetry: A survey," *Comput. Netw.*, vol. 186, Feb. 2021, Art. no. 107763.

Azza H. Ahmed (Member, IEEE) received the master's degree from the University of Nottingham in 2012. She is currently pursuing the Ph.D. degree with the Simula Metropolitan Center for Digital Engineering, Oslo, Norway. Her research interests include communication networks management and control, network performance optimization, network automation, and machine learning to solve networks problems.

Ahmed Elmokashfi received the Ph.D. degree from the University of Oslo in 2011. He is a Research Professor with the Simula Metropolitan Center for Digital Engineering in Norway. He is currently working as the Head of the Center for Resilient Networks and Applications, which is part of the Simula Metropolitan Center, which is funded by the Norwegian Ministry of Transport and Communication. In particular, he focused on studying resilience, scalability, and evolution of the Internet infrastructure; the measurement and quantification of robustness in mobile broadband networks; and the understanding of dynamical complex systems. Over the past few years, he has been leading and contributing to the development, operation and management of the NorNet testbed infrastructure, which is a countrywide measurement setup for monitoring the performance of mobile broadband networks in Norway. His research interests lie in network measurements and performance.