SeqDQN: Multi-Agent Deep Reinforcement Learning for Uplink URLLC with Strict Deadlines

Benoît-Marie Robaglia
and Marceau Coupechoux
LTCI, Telecom Paris
Institut Polytechnique de Paris
Email: robaglia@telecom-paris.fr

Dimitrios Tsilimantos
and Apostolos Destounis
Advanced Wireless Technology Lab
Paris Research Center
Huawei Technologies Co. Ltd.

***Abstract*—Recent studies suggest that Multi-Agent Reinforcement Learning (MARL) can be a promising approach to tackle wireless telecommunication problems and Multiple Access (MA) in particular. The most relevant MARL algorithms for distributed MA are those with "decentralized execution", where an agent's actions are only functions of their own local observation history and agents cannot exchange any information. Centralized-Training-Decentralized-Execution (CTDE) and Independent Learning (IL) are the two main families in this category. However, while the former suffers from high communication overhead during the centralized training, the latter suffers from various theoretical shortcomings. In this paper, we first study the performance of these two MARL frameworks in the context of Ultra Reliable Low Latency Communication (URLLC), where MA is constrained by strict deadlines. Second, we propose a new distributed MARL framework, namely SeqDQN, leveraging the constraints of our URLLC problem to train agents in a more efficient way. We demonstrate that not only does our solution outperform the traditional random access baselines, but it also outperforms state-of-the-art MARL algorithms in terms of performance and convergence time.**

***Index Terms*—Distributed Multiple Access, Deep Multi-Agent Reinforcement Learning, Internet of Things, Wireless sensor networks, URLLC.**

I. Introduction

The next generation wireless networks need to address high-performance use cases related to industrial automation [1] with the Internet-of-Things (IoT). A few examples are factory automation, motion control and vehicular networks. For instance, IoT sensors can monitor a factory and can be required to communicate an emergency event as fast as possible so that devices can react accordingly. Therefore, it is necessary that the communication of these extreme events is done reliably and in a very short delay. This type of communication is called *Ultra-Reliable Low-Latency Communication* (URLLC) [2]. However, traditional protocols fail to meet these requirements. The authors in [3] highlight the various sources of latency in cellular networks, the random access delay between a user and the Base Station (BS) being a major one (which can go up to 10 ms). Grant-free random access protocols such as Slotted ALOHA [4] can reduce this kind of delay, but the latency exponentially increases with the number of users, which makes them incapable of dealing with the URLLC constraint.

A new generation of "intelligent algorithms", leveraging the latest advances in Machine Learning and more particularly in Reinforcement Learning (RL) has emerged to tackle the URLLC constraint for MA in IoT [5]. In the same direction, the objective of this paper is to tackle a fully distributed MA problem with the URLLC constraint by applying Multi-Agent RL (MARL). We propose SeqDQN, a distributed algorithm that outperforms MA baselines and the state-of-the-art MARL algorithms for URLLC communications with strict deadlines.

A. Related Work

RL is a branch of Machine Learning where agents learn a policy by interacting with an environment [6]. The development of Deep Neural Networks (DNN) allowed RL to solve high dimensional Markov Decision Processes (MDPs) [6] and thus, to explore their potential use in wireless communications and MA in particular [7]. The most natural way of applying single-agent RL to MA is to model the BS with an RL algorithm that decides which device to schedule at every slot. The main issue in this case is the heavy communication between the BS and the devices as the RL agent needs to know if a device has a packet to transmit before scheduling it. A way to alleviate this issue is proposed by [8], where the authors propose to reduce the communication overhead by assuming partial observability and learning the packet arrival distribution with a Recurrent Neural Network (RNN). However, this approach is not always appropriate as the BS is not aware of the packet arrivals and may miss transmission opportunities causing large delays and packet losses. Another promising RL approach to address the MA problem is MARL. MARL is a paradigm where at least two agents concurrently interact with an environment. The main MARL frameworks that have been applied to a MA problem are:

Independent Learning (IL): introduced by [9], IL is the most straightforward way to expand the RL framework to multiple agents. The idea is to equip each agent in the MARL problem with a single-agent RL algorithm, considering the actions of the other agents as part of the environment. Even if this approach has the benefits of being fully distributed and decentralized by construction, it suffers from various theoretical limitations that can result in instabilities during training and convergence to suboptimal policies [9]. The main shortcoming of this method is the non-stationarity from the perspective of one agent created by the actions of the others.

One of the first algorithms used is independent Q-learning with DNNs, namely independent DQN (iDQN) [6]. Because of its ability to train fully decentralized agents, IL is the most used framework in MA. For instance, [10]–[12] apply IL to Dynamic Spectrum Access (DSA), where secondary users use RL to learn a transmission protocol. Yet, none of these papers take into consideration the theoretical shortcomings mentioned earlier. Furthermore, no previous work has applied IL to tackle a URLLC problem on the uplink with strict deadlines.

Centralized Training with Decentralized Execution (CTDE): recent research has been focusing on addressing the shortcomings of IL. One of the most common ways of achieving this is to use a centralized critic during training, taking advantage of the fact that a lot of MARL problems can be trained in a centralized way (in a factory for example) before being deployed in the real world. This approach reduces or removes the issues of non-stationarity and partial observability raised by IL, while still learning decentralized policies. One of the main frameworks using CTDE is called value factorisation. For cooperative problems (assuming a single collective reward), value factorisation aims to decompose the joint state-action value function into individual observation-action value functions. Two of the most famous algorithms are Value Decomposition Network (VDN) [13] that learns a linear decomposition of the joint Q-function and QMIX [14] that learns a more complex monotonic factorisation of it, and has demonstrated better performance. The authors of [15] justify the use of the centralized training by leveraging mobile edge computing. They apply the QMIX algorithm to tackle the DSA problem, modelling it as a Decentralized-POMDP (Dec-POMDP) problem [16]. In spite of the potential of CTDE for MA, it has not been applied to a problem with a strict latency constraint and the way to train a centralized critic remains an issue for the IoT because of the high communication overhead.

Based on this literature study, MARL methods have not been applied to a URLLC problem where the communication is on the uplink and with strict deadlines. The performance of IL and CTDE thus needs to be assessed in this context. In addition, both frameworks have strengths and weaknesses that led us to formulate our own algorithm for MA: SeqDQN.

B. Our contribution

In this paper, we formulate a URLLC problem by considering heterogeneous devices that need to transmit short packets on the uplink to a BS within a given strict deadline. We equip each device with a deep MARL algorithm in order to learn a transmission protocol by interacting with the other devices and the environment. We propose SeqDQN, a distributed MARL algorithm where agents do not update their Q-functions simultaneously. Instead, they update their Q-function sequentially, starting with the devices with the most stringent latency requirement. The advantages of this method are: 1) We reduce the non-stationarity caused by multiple agents learning concurrently, which is a major drawback of IL, 2) our proposed method is more scalable to a large number of agents than CTDE and 3) training is much faster than the existing MARL algorithms (iDQN and QMIX). We show that SeqDQN outperforms both the standard MA baselines and MARL algorithms on the traffic models we consider.

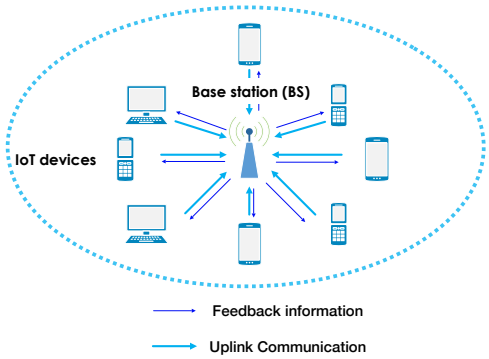


Fig. 1: System model with heterogeneous devices.

II. Problem formulation

A. System model

We consider a network of N devices communicating with a BS over a wireless shared channel on the uplink (Fig. 1). Time is slotted and at every slot, devices can choose to transmit a packet or to remain idle. All packets are supposed to require the same transmission time of one time slot and the propagation delay is assumed to be negligible. Moreover, each device has an individual air interface latency constraint δ_i, such that a packet is dropped if it has not been transmitted within δ_i slots after its arrival in the buffer. We assume slot synchronization, i.e. all devices are aligned on a common start of each slot. Finally, we consider a collision channel model: if a collision occurs, the packets are not delivered to the BS but can be re-transmitted until their deadline is reached. After each slot, the users have access to what happened in the slot, i.e., whether a transmission was successful, the channel was idle or a collision occurred. This information is obtained by a feedback signal (ACK/NACK) broadcast from the BS to all the devices. In order to evaluate the different MARL algorithms, we define a *URLLC score* as the number of packets delivered before expiry divided by the number of generated packets.

B. Traffic model

We study the performance of MARL using 2 traffic models based on the framework of [17] and the 3GPP [18].

Probabilistic periodic traffic: Inspired by [17], we first consider a framework where the traffic pattern of every device is periodic, i.e., every period of T_i time slots, device i receives a packet with probability p_i. We allow the devices to be heterogeneous in the sense that they can have different packet arrival probabilities p_i, different periods T_i and they are not synchronous: Each device is assigned an offset parameter $f_i \in [0, T_i]$, so that packet arrivals can occur only at time instants $f_i + mT_i$, where m is an integer. At every slot $t \geq 0$,

the probability for a device $i \in [1, N]$ of having a new packet is $q_i(t|f_i, p_i, T_i) = \mathbb{1}_{\{t[T_i]=f_i\}} p_i$, where $\mathbb{1}_{\{\cdot\}}$ is the indicator function and $[\cdot]$ is the modulo operator.

Deterministic periodic traffic: Defined in [18, Annex A], the deterministic periodic traffic is a special case of the previous framework with all arrival probabilities equal to 1. The main challenge of this framework is for the devices to learn the optimal schedule. This optimal schedule can be used subsequently by a contention-free grant-free access algorithm as standardized in 3GPP Release R15 for URLLC [18]. In the probabilistic and deterministic periodic traffic models, we assume that $\delta_i \leq T_i$ for all i and thus all devices have a one-packet buffer.

C. Decentralized Partially Observable Markov Decision Process (Dec-POMDP)

We formulate our URLLC problem as a Dec-POMDP [16]. A Dec-POMDP is a cooperative stochastic team game with partial observability. This framework can be regarded as an extension, for multiple agents, of an MDP where a team of agents collaborate in order to maximize the same objective. Besides, it is partially observable, meaning that agents cannot see the full state and take actions based on their own observations.

In our case, let $\mathbb{S} = \mathbb{S}_1 \times \mathbb{S}_2 \times \cdots \times \mathbb{S}_N$ be the set of environmental states where $\boldsymbol{s_t} = (s_t^1, s_t^2, \dots s_t^N) \in \mathbb{S}$ is the concatenation of all individual states. The devices have a buffer of size 1 and the local state of a device i at slot t is $s_t^i = (d_t^i, c_{t-1})$, where $d_t^i \in \mathbb{N}$ is the time in number of slots that the packet has already spent in the buffer and c_{t-1} is the last feedback from the BS. We set $c_{t-1} = 1$ if at $t-1$ a packet was successfully transmitted, -1 if a collision occurred and 0 if the channel was idle.

A local action of a device i is $a_t^i \in \mathbb{A}^i = \{0, 1\}$, where $a_t^i = 1$ if the device transmits and 0 otherwise. A device i makes an action according to a policy function $\pi^i : \mathbb{S}^i \to \Delta(\mathbb{A}^i)$. This function can be probabilistic or deterministic. We use the index $-i$ to refer to all agents different from i. For example, the joint action $\boldsymbol{a} = (a^1, a^2, \dots, a^n) \in \mathbb{A}$ can be written $\boldsymbol{a} = (a^i, a^{-i})$.

We now specify the reward function $\mathcal{R} : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$. In a Dec-POMDP, agents have identical interests, which means that they have the same reward function. Users collectively want to maximize the URLLC score by maximizing the number of successful transmissions. At each slot t, agents get as reward $r_t = \mathcal{R}(\boldsymbol{s_t}, \boldsymbol{a_t})$ the ACK/NACK feedback from the BS ($+1$ if a packet is successfully transmitted, -1 if a collision occurred, 0 if the channel was idle). In other words, $r_t = c_t$. The objective for each agent i is to find a policy π^i that maximizes the expected cumulative discounted reward over a finite horizon T:

$$\mathbb{E}_{\boldsymbol{s_{t+1}} \sim \mathbb{P}, a^{-i} \sim \pi^{-i}} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(\boldsymbol{s_t}, (a_t^i, a_t^{-i}) | a_t^i \sim \pi^i(\cdot|s_t^i), \boldsymbol{s_0} \right] \tag{1}$$

with $\gamma \in (0, 1]$ the discount factor that allows the agents to balance immediate rewards with future ones, and $\mathbb{P} : \mathbb{S} \times \mathbb{A} \mapsto \mathbb{S}$ the transition function. In our problem, we consider a finite horizon Dec-POMDP where an episode is of length T slots.

Note that it is also possible to model this system with a more general framework where agents have individual interests. However, we have chosen the Dec-POMDP framework as: 1) The main CTDE algorithms have been designed for Dec-POMDP only [13], [14], [19]; 2) We experimentally have observed that the performance of IL algorithms with our system model is very similar whether we use an individual or a collective reward.

III. Algorithms

Among the existing MARL algorithms, we restrict our approach to the ones with "decentralized execution" where an agent can only choose an action based on its local observation.

A. Independent DQN (iDQN)

Deep Q-learning is one of the most used framework in RL. It aims at learning a state-action value function $Q(s, a)$ with a neural network. A Q-learning agent then selects the action that maximizes the Q-function, i.e. $\pi(a|s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$. In Deep Q-Networks (DQN) [6], an agent stores the system transitions (s, a, r, s') in a replay buffer where the agent observes the next state s' after taking the action a in the state s and receiving the reward r. The neural network's parameters θ are learnt by sampling batches of b transitions from the replay buffer and minimizing the following loss, called TD-error:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[y_i - Q(s_i, a_i; \theta) \right]^2 \tag{2}$$

where $y_i = r_i + \gamma \max_{a'} Q(s_i', a'; \theta^-)$ is the TD target and θ^- are the parameters of what the authors call the *target network* that are used to stabilize the training procedure. These parameters are an old version of the parameters θ and are periodically updated.

The most natural way to adapt DQN to a multi-agent scenario is independent Q-learning [9]. The idea is to decompose a n-agent problem in n single agent problems, where each agent considers the others as part of the environment. When users model their Q-function with a DNN, we get iDQN.

iDQN has the advantage of being a fully distributed algorithm in the sense that training and execution are decentralized, which makes it a good candidate for a distributed MA problem. However, this framework does not address the non-stationarity caused by the learning of other agents and thus has no convergence guarantees.

In order to be fair in the comparison with the other MARL algorithms, we equip each Q-network with a Gated Recurrent Unit (GRU) layer [20]. Thus, the local state s_t^i of an agent i at step t is replaced by the action-observation history $h_t^i = (s_t^i, s_{t-1}^i, a_{t-1}^i, \dots, s_0^i, a_0^i)$. This way of handling partial observability in Deep RL is described in [21].

Moreover, we noticed that without the use of a recurrent architecture, the iDQN algorithm fails to converge when the number of devices increases.

B. QMIX

QMIX [14] is a value decomposition algorithm that relies on the factorization of the global Q-function Q_{tot} by individual Q-functions Q_n to enable distributed execution. In practice, each agent estimates a Q-value with a neural network that is going to be passed through a neural network called the mixing network. It combines all these individual Q-values into a joint Q-value $Q_{tot} = \texttt{Mixing\_Network}(Q_1, Q_2, \ldots, Q_n)$. The latter is used to minimize the DQN loss (2) and the gradient is backpropagated to the individual Q-functions. In practice, we want to factorize the total Q-value such that maximizing the total Q-value gives the same result as maximizing the individual Q-values:

$$\arg\max_{a} Q_{tot}(\boldsymbol{h}, \boldsymbol{a}) = \left[\arg\max_{a^n} Q_n(h^n, a^n)\right]_{n=1,\ldots,N} \quad (3)$$

To do so, the mixing network enforces a monotonic constraint between Q_{tot} and Q_n:

$$\frac{\partial Q_{tot}}{\partial Q_n} \quad \geq \quad 0, \ \ \forall n \quad (4)$$

which is fulfilled by constraining the parameters of the mixing network to be positive. The advantage of this approach is that it tackles the non-stationarity issue during training thanks to the centralized Q-function. However, the communication overhead during the training phase may be a problem for IoT scenarios when we increase the number of devices.

C. Sequential DQN (SeqDQN)

Our proposed algorithm, called SeqDQN, is inspired by the idea of two-timescale training [22] which tries to take advantage of iDQN without the non-stationarity issue. In SeqDQN, n DQN agents update their Q-function sequentially on different exploration phases (EP). During one EP, only one agent updates its policy while the others take actions according to their last learnt policy. Thus, we remove the non-stationarity caused by the other learning decision makers and each agent solves a POMDP problem during each EP in order to learn a best response to the others. The authors of [22] show that this methodology provides a decentralized Q-learning framework where algorithms converge to equilibrium policies almost surely in fully observable weakly acyclic games.

To tackle our MA problem, we combine the two-timescale training with three elements. First, we use a GRU [20] to address partial observability in POMDP [21]. Second, we create clusters of devices with the same deadline. Clusters are then trained sequentially from the smallest deadline to the largest one. When a cluster is trained, the users it includes update their policy one after the other until convergence. Third, devices that have not been trained so far do not transmit to remove stochasticity during the EP and speed up the learning.

The pseudo-code of SeqDQN is shown in Algorithm 1. Inside a cluster, agents are trained sequentially K cycles. In an EP, when the policy of the learning agent has not improved in L episodes, we consider that the agent has converged and end the EP. We set the Q-function of the agent to the best one learnt during this EP in terms of total reward. We train the clusters J rounds so that the first trained clusters can adjust their policies to the ones with a larger deadline. At each round, the learning rate is decreased by a factor α. In practice, we set up the training sequence and clusters of users once at the beginning of the experiment. The BS groups the users by deadline and assigns them an ID corresponding to an EP where a device is allowed to update its Q-network. When an EP is finished, the BS communicates what device starts its EP via the feedback signal. There is no additional information exchange during an episode. The sequential training is a pre-processing set up that does not need to be repeated once the network is in operation. During the network operation, the decisions are fully decentralized.

Algorithm 1: SeqDQN for distributed multiple access

```
Initialize the Q-networks Q_1, Q_2, ..., Q_n;
Cluster the users in subsets C_1, C_2, ... C_d with users
  in C_i having a deadline δ_i such that δ_i < δ_{i+1} ∀i.
for j = 1, 2, ..., J do
    η ← η × α
    for i = 1, 2 ..., d do
        if i = 1 then
            Set the policies of clusters c > i to not
              transmit.
        Fix the policies of clusters c < i to their
          current policy.
        for cycle = 1, 2, ..., K do
            /* Sequentially train the users of C_i
               K times                                */
            for m ∈ C_i do
                /* EP of agent m                     */
                Run DQN for agent m and fix π^{-m}.
                Update Q^m with (2) and learning rate η.
                if π^m has not improved in L episodes
                  then
                    End EP
                    Set the policy of π^m to the best one.
```

D. Baselines

We compare the above algorithms to two baselines:

• **Contention-based grant-free access (GF) [23]**: All devices with a packet to transmit access the channel with the same probability p. We optimize the transmission probability p experimentally for every number of agents such that the URLLC score is maximised. We consider the reactive scheme so that when a device receives a NACK feedback from the BS after transmitting, it will re-transmit the same packet with probability p until the reception of an ACK feedback or the expiry of the packet.

• **Round Robin scheduler (RR)**: This algorithm schedules devices in a cycle, so that the time resource is fairly shared between them. Devices are ranked in ascending order with deadlines plus offsets.

TABLE I: Parameters of the traffic models

Parameters	**Probabilistic Dense**	**Probabilistic Sparse**	**Deterministic**
Arrival Probabilities	$\mathcal{U}\{0.2, 0.4, 0.6, 0.8\}$	$\mathcal{U}\{0.05, 0.1\}$	1
Deadlines	$\mathcal{U}\{5, 10, 15, 20\}$	$\mathcal{U}\{2, 4\}$	$\mathcal{U}\{4, 6, 8, 10\}$
Periods (T_i)	$\mathcal{U}\{10, 20\}$	$\mathcal{U}\{5, 10\}$	$\mathcal{U}\{8, 10\}$
Offsets	$\mathcal{U}[0, 4]$	$\mathcal{U}[0, 4]$	$\mathcal{U}[0, 4]$

IV. Experiments

A. Simulation settings

We consider the three different settings for our experiments: a *dense probabilistic periodic traffic*, a *sparse probabilistic periodic traffic* and a *deterministic periodic traffic* from 4 to 28 users depending on the scenario. The parameters of these traffic models are given in Table I and the parameters of the MARL algorithms are given in Table II. These numbers of devices are in line with the 3GPP recommendations for URLLC [18], where the use cases consider up to 10 users per cell. For every traffic model, periods T_i are chosen uniformly under the condition $\delta_i < T_i$. In Table I, $\mathcal{U}$ designates the uniform distribution over a finite set or an interval. The hyper parameters have been optimized with grid search.

TABLE II: Parameters of the Q-learning algorithms

Parameter	Value
Discount factor (γ)	0.9
Initial learning rate (η)	10^{-3}
Learning rate decrease factor (α)	0.2
Batch size	128
History length	20
Episode length (T)	200 slots
Training length	50k episodes
Final exploration rate (ϵ)	0.1
Number of cycles (K)	5
Convergence criterion (L)	300
Training rounds for clusters (J)	3
Q-network architecture	1 GRU layer + 3 Linear layers
Activation functions	ReLU
Linear layer	100 neurons
GRU layer	100 neurons
Number of seeds	5

B. Convergence speed of MARL algorithms

We analyze in Fig. 2 the evolution of the performance of the three MARL algorithms during the training phase for the dense probabilistic periodic traffic with 12 devices. The conclusions are similar for other traffic models and numbers of devices. Instead of comparing the algorithms as a function of the number of episodes, we present results as a function of the number of gradient updates in the x-axis. The reason is that iDQN and QMIX perform as many updates as the number of devices in every episode, whereas SeqDQN trains a single device. The number of gradient updates is thus more representative of the convergence speed.

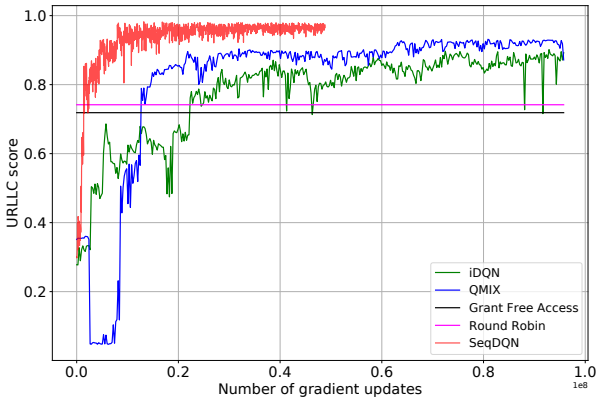
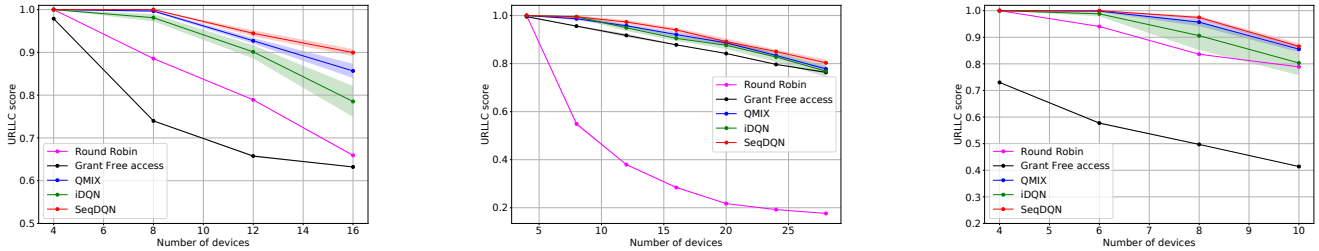


Fig. 2: Evolution of the URLLC score during the training for the dense probabilistic periodic traffic with 12 devices.

First, we can see that SeqDQN manages to reach a better optimum at the end of the training. Second, we can see that it is the fastest algorithm to converge. This can be explained by two arguments. When a cluster of devices is being trained, the devices with a higher deadline are inactive. Therefore, the stochasticity of the environment is removed so it is easier for the learning devices to converge. Moreover, as only one agent explores at the same time, exploration is not affected by the noise caused by the concurrent learning of other agents as it is the case for iDQN and QMIX. Third, we observe that the training of SeqDQN has more variance than QMIX's. This can be explained by the fact that training is not centralized and devices are not trained simultaneously. Additionally, when a new agent starts its EP, it needs to adapt to the new policies of the other devices which creates variance. Reducing the learning rate at every round helps mitigating this effect. Finally, we notice that QMIX outperforms iDQN in terms of convergence speed and optimum attained, which is expected as centralized training is supposed to encourage cooperation.

C. URLLC score

We run simulations with a dense probabilistic periodic traffic (Fig.3a), a sparse probabilistic periodic traffic (Fig.3b) and a periodic deterministic traffic (Fig. 3c). We show the evolution of the URLLC score as a function of the number of devices. First, it is clear that the MARL algorithms outperform the MA baselines (GF and RR) in every scenario. We observe that the GF protocol performs very poorly in dense settings when the load is high (Fig 3a and Fig 3c) while it performs very well in a sporadic traffic when the load is lower (Fig 3b). Indeed, the larger the number of packets to transmit, the larger the number of collisions, thus the lower the performance of the GF protocol. On the contrary, we can see in Fig 3a and Fig 3c that the RR scheduler performs better when the load is high as 1) it schedules one packet at a time and thus avoids



(a) Dense periodic probabilistic traffic. (b) Sparse periodic probabilistic traffic (c) Periodic deterministic traffic.

Fig. 3: Evolution of the URLLC score with respect to the number of devices for various traffic models.

collisions; 2) the probability of scheduling a user with a packet is higher. Nevertheless, it performs very poorly in the sparse periodic traffic (Fig 3b) as it is not aware when a device has a packet to transmit. Furthermore, we can notice that in all three scenarios, SeqDQN not only outperforms iDQN and QMIX, but it also has less variance when we change the initialization of the neural networks. This can be explained by the fact that in SeqDQN, only one agent explores at a time whereas in iDQN and QMIX, all agents explore concurrently which makes an equilibrium harder to reach. We also observe that the variance of iDQN increases with the number of devices. Indeed, positive rewards become quite sparse as they are obtained when one device is active while all other ones remain idle. This problem is known in the literature on MARL, see e.g. [19]. Finally, we note that in some cases, QMIX's performance is similar to SeqDQN's. The centralized training that encourages agent coordination can explain this. However, it necessitates a lot of communication during training, whereas SeqDQN avoids this problem because training is decentralized.

V. Conclusion

In this paper, we consider an uplink multiple access problem with strict deadlines. We study the most commonly MARL approaches to learn a transmission protocol in this URLLC context. We assess the performance of iDQN and QMIX and propose a new distributed algorithm, called SeqDQN, which addresses the issues of scalability and non-stationarity caused by the other learning agents. We show that SeqDQN has three major advantages compared to the existing MARL algorithms: 1) It can reach a better operating point when we allow the devices with the most time-critical information to learn a transmission protocol first, 2) Training the agents one at a time significantly speeds up the training time, 3) Training agents progressively and fixing the policies of the already trained agents allow us to handle a larger number of devices.

References

[1] G. Brown *et al.*, "Ultra-reliable low-latency 5g for industrial automation," *Technol. Rep. Qualcomm*, vol. 2, p. 52065394, 2018.

[2] J. J. Nielsen, R. Liu, and P. Popovski, "Ultra-reliable low latency communication using interface diversity," *IEEE Trans. Commun.*, vol. 66, no. 3, pp. 1322–1334, 2017.

[3] H. Chen *et al.*, "Ultra-reliable low latency cellular networks: Use cases, challenges and approaches," *IEEE Commun. Mag.*, vol. 56, no. 12, pp. 119–125, 2018.

[4] L. G. Roberts, "ALOHA packet system with and without slots and capture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 5, no. 2, pp. 28–42, 1975.

[5] M. A. Al-Garadi *et al.*, "A survey of machine and deep learning methods for internet of things (IoT) security," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 1646–1685, 2020.

[6] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] 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, 2021.

[8] B.-M. Robaglia, A. Destounis, M. Coupechoux, and D. Tsilimantos, "Deep Reinforcement Learning for Scheduling Uplink IoT Traffic with Strict Deadlines," in *IEEE GLOBECOM*, 2021.

[9] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *ICML*, 1993.

[10] H.-H. Chang *et al.*, "Distributive dynamic spectrum access through deep reinforcement learning: A reservoir computing-based approach," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1938–1948, 2018.

[11] Y. Xu, J. Yu, and R. M. Buehrer, "The application of deep reinforcement learning to distributed spectrum access in dynamic heterogeneous environments with partial observations," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4494–4506, 2020.

[12] Y. Xu, J. Yu, W. C. Headley, and R. M. Buehrer, "Deep reinforcement learning for dynamic spectrum access in wireless networks," in *IEEE MILCOM*, 2018.

[13] P. Sunehag *et al.*, "Value-decomposition networks for cooperative multi-agent learning," in *AAMAS*, 2018.

[14] T. Rashid *et al.*, "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning," in *ICML*, 2018.

[15] X. Tan *et al.*, "Cooperative multi-agent reinforcement learning based distributed dynamic spectrum access in cognitive radio networks," *arXiv:2106.09274*, 2021.

[16] F. A. Oliehoek, *Decentralized POMDPs*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 471–503.

[17] I.-H. Hou and P. R. Kumar, "Packets with deadlines: A framework for real-time wireless networks," *Synth. Lect. Commun*, vol. 6, no. 1, pp. 1–116, 2013.

[18] 3GPP, "Study on physical layer enhancements for NR ultra-reliable and low latency case (URLLC)," 3rd Generation Partnership Project (3GPP), TR 38.824. [Online]. Available: http://www.3gpp.org/DynaReport/38824.htm

[19] R. Lowe *et al.*, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *NeurIPS*, 2017.

[20] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014.

[21] M. Hausknecht and P. Stone, "Deep Recurrent Q-learning for Partially Observable MDPs," in *AAAI Fall Symposeum*, 2015.

[22] G. Arslan and S. Yüksel, "Decentralized Q-Learning for Stochastic Teams and Games," *IEEE Trans. Autom. Control*, vol. 62, no. 4, pp. 1545–1558, 2017.

[23] N. H. Mahmood, R. Abreu, R. Böhnke, M. Schubert, G. Berardinelli, and T. H. Jacobsen, "Uplink grant-free access solutions for urllc services in 5g new radio," in *2019 16th International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, 2019, pp. 607–612.