2022 IEEE 96th Vehicular Technology Conference (VTC2022-Fall) | 978-1-6654-5468-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/VTC2022-Fall57202.2022.10012730

TinyQMIX: Distributed Access Control for mMTC via Multi-agent Reinforcement Learning

Tien Thanh Le*†, Yusheng Ji†*, John C.S. Lui ‡
*Department of Informatics, The Graduate University for Advanced Studies, SOKENDAI, Tokyo, Japan
†National Institute of Informatics, Tokyo, Japan
‡Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong
Email: *†{lethanh, kei}@nii.ac.jp;‡cslui@cse.cuhk.edu.hk

***Abstract*—Distributed access control is a crucial component for massive machine type communication (mMTC). In this communication scenario, centralized resource allocation is not scalable because resource configurations have to be sent frequently from the base station to a massive number of devices. We investigate distributed reinforcement learning for resource selection without relying on centralized control. Another important feature of mMTC is the sporadic and dynamic change of traffic. Existing studies on distributed access control assume that traffic load is static or they are able to gradually adapt to the dynamic traffic. We minimize the adaptation period by training TinyQMIX, which is a lightweight multi-agent deep reinforcement learning model, to learn a distributed wireless resource selection policy under various traffic patterns before deployment. Therefore, the trained agents are able to quickly adapt to dynamic traffic and provide low access delay. Numerical results are presented to support our claims.**

***Index Terms*—mMTC, multi agent deep reinforcement learning**

I. Introduction

In recent years, the number of connected devices has grown exponentially due to the proliferation of Internet of Things (IoT) applications. Many IoT applications are enabled by massive machine type communication (mMTC), i.e, autonomous vehicles, industrial automation, or environmental sensing. It has been projected that about half of total global connections (about 15 billion devices) will be mMTC devices [1].

The traffic features of mMTC are inherently different from other communication scenarios. First, mMTC protocols should support a *high overloading factor*, in which a large number of devices share a small number of wireless resources. Although the number of wireless resources is small, the limited resource would still be able to support mMTC because mMTC devices typically transmit a *low volume of short packets and uplink data* [2]. Another important feature of mMTC is *sporadic traffic* [3]. Sporadic traffic means that devices do not always have data to transmit and only a random subset of devices access the network in each timeslot. Moreover, mMTC traffic tends to be *dynamic*, which means that the amount of data generated and sent by mMTC devices would unlikely be a constant rate throughout. For example, IoT sensors normally send regular status updates to the server, but sometime an important event would be triggered, so these devices must send a larger amount of information regarding that important event [2]. In short, future mMTC protocols are expected to address the mentioned features, by allowing high overloading, and supporting sporadic, and dynamic uplink traffic.

To solve this problem, many of approaches have been put forward in 5G's multiple access control (MAC) layer such as uplink contention-based grant-free (GF)-non-orthogonal multiple access (NOMA) [4]. NOMA would increase the system overloading because it enables multiple devices to reuse the same time-frequency resource via power domain or code domain multiplexing. The contention-based grant-free mechanism reduces the delay of sporadically arrived packets since it allows devices to transmit data directly to base station (BS) without the need for the request and grant procedure.

A growing body of work has improved the original contention-based GF NOMA by equipping the protocol with reinforcement learning. In general, reinforcement learning techniques can solve the network optimization problem in a data-driven manner instead of relying on the analytical network model. For the current problem, reinforcement learning is adopted for selecting the MAC layer's parameters. In [5], Deep Q Networks (DQN) with long short-term memory (LSTM) architecture are deployed at each mMTC device to select the wireless resource and power level to maximize the network throughput. This is an independent DQN policy. However, for independent learning methods, the changing policy of one agent leads to changing the optimal target policy of another agent. Thus, *independent DQN has no convergence guarantees and may lead to poor performance* [6, Sec 3.3].

The problem of independent DQN can be mitigated by multi-agent deep reinforcement learning (MADRL) policies such as Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning (QMIX) [6]. In QMIX, the LSTM Q-function of all devices are trained together in a centralized simulator, such that the Q-value of an agent is consistent with the joint-action's Q-value function of all agents. After training, agents can select the best joint-action independently without any communication. Huang, Wong, and Schober leveraged QMIX for selecting pilot sequences [7], while Guo, Chen, Liu, *et al.* also applied QMIX for deciding whether to back-off in 802.11 network [8]. Both proposed QMIX policies demonstrated their superiority to independent DQN and other heuristic policies. However, the main drawback of these policies is *the computational complexity of their LSTM architecture*. In fact, running this neural network's

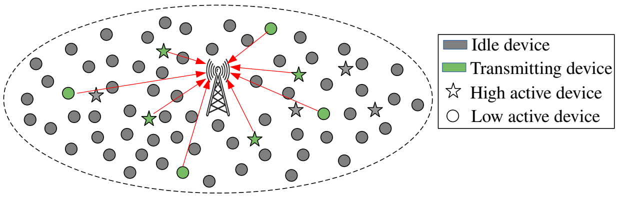


Fig. 1: mMTC devices uplink transmission model

architecture on IoT devices take a long time while the deadline for decision-making is short. Also, the problem of *low rate and sporadic traffic were not explicitly addressed*.

Sporadic traffic for uplink contention-based NOMA has been investigated in [9]. two distributed Q-learning (DQL) schemes denoted as ADDQ and PDDQ to select resources that minimize collisions. However, they only consider that *traffic intensity of each device remains constant throughout its lifetime*.

To the best of our knowledge, there is no prior work on a fully distributed MAC protocol that addresses the wireless resource selection problem with sporadic and dynamic traffic in mMTC. Therefore, the goal of this work is to design a new distributed resource selection protocol to solve this problem. The primary contributions of this paper are as follows:

- We map the problem into a decentralized partial observable Markov decision process (Dec-POMDP). We also design a lightweight local observation and MADRL policy called TinyQMIX. TinyQMIX agents have a low model complexity, such that they can be implemented on mMTC devices
- TinyQMIX policy is trained on a wide variety of sporadic traffic scenarios, allowing the trained agents to quickly adapt to the ever-changing traffic dynamic. Devices can independently select the best wireless resource to minimize system-wide access delay.
- Numerical simulations are performed to evaluate the transmission delay of the TinyQMIX with other widely-adopted heuristic and MADRL algorithms.

The rest of this paper is organized as follows. Section II presents the network and traffic model. We propose our Dec-POMDP formulation and TinyQMIX for dynamic access control in mMTC in the Section III. Numerical simulation is presented in Section IV

II. System Model

Here, we present the scope of our study, which includes the mMTC network model, the dynamic and sporadic traffic model, and the dynamic access control model.

A. Network scenario

For the ease of presentation, we first consider a single-cell wireless system in Figure 1. The BS is located at the center and is surrounded by N devices within the coverage radius. Let $\mathcal{N} = \{1, 2, \dots, i, \dots, N\}$ be the set of N devices. Assume that the system is slotted and time synchronized. Because the mMTC traffic contains mostly short packets, we assume that the service time of one packet is one timeslot.

B. Traffic model

Assume that the packet arrival rate per timeslot for the i^{th} device is $\lambda_i(t)$ where t is the index of a timeslot. Similar to [9], we assume the set of devices $\mathcal{N}$ can be divided into two groups: high active devices $\mathcal{N}_h$ and low active devices $\mathcal{N}_l$. Also, the network is likely to contain more low active devices ($|\mathcal{N}_h| \ll |\mathcal{N}_l|$).

We consider the dynamic traffic arrival scenario: The distribution of traffic of every device changes within their lifetime. Therefore, the parameter of packet arrival distribution $\lambda(t)$ is a time-dependent variable. We assume that devices have the same dynamic within an interval ΔT, where ΔT is a constant. All devices change their arrival distribution $\lambda(t)$ synchronously after a constant ΔT timeslots. We like to note that since our proposed method is data-driven, it would also work with asynchronous changes.

C. Distributed access control model

Let the total duration of τ timeslots be the scheduling interval. At the beginning of every scheduling interval, each device selects any of C resource units to send data. Thus, the scheduling decision space is C^N, which grows exponentially with the number of devices and resources. We reduce the scheduling space by dividing N devices into many groups, each containing $N^{'}$ devices and M resource units, similar to what has been done in [5], [9], [10]. Devices can be grouped arbitrarily as long as less than N' devices share M resources. Let $\phi = \frac{N^{'}}{M}$ be the overloading factor, and $\phi \gg 1$.

A collision occurs when two or more devices select the same resource unit and transmit their data in the same timeslot. Whenever this happens, each colliding device follows a binary exponential back-off procedure to resolve the contention. Particularly, each device retains a contention window value cw, which is initialized at 1. If there is a collision, the contention window is doubled until it reaches $CW_{\max}$. Then, each device draws a uniform random integer $r \sim U([1, cw])$, and waits for r timeslots before resuming the transmission. To counter the effect of sporadic traffic, each device can temporarily store a maximum of L_{buffer} packets in its buffer. Also, a collided packet can be retried for a maximum of L_{retry} times. By doing that, the fraction of dropped packets is negligible, and the high-reliability requirement could be realized.

The distributed access control mechanism here is a general abstraction and it can be integrated with contention-based GF-NOMA. A resource unit in GF-NOMA can be a tuple of frequency, pilot sequence, and NOMA code-book. We suppose that the given system adopts time duplex division (TDD) mode. Pilot sequences are broadcasted by BS, then devices measure channel state information (CSI) to calibrate their power level to satisfy the receiving power requirement. Assume that devices are able to satisfy the requirement, so the only cause for transmission failure is transmission collision.

III. Distributed Dynamic Resource Selection - TinyQMIX

The distributed wireless resource selection problem can be formulated as a Dec-POMDP [11], which is defined as a tuple $G = \langle N', S, U, P, r, Z, \gamma \rangle$. The system consists of N' agents (or N' devices). Each agent runs according to Algorithm 1 with TinyQMIX decision-making capability combined with random access procedure. $s \in S$ is the true state of the environment. The state can either be: (1) the state of the network simulation's program when the agents are trained, (2) the state of the BS and channel quality at deployment time.

Algorithm 1 TinyQMIX agent i for distributed access control

```
Input: Agent's DNN parameter $\theta^i$, MAC's parameter
    $CW_{\max}, L_{\text{buffer}}, L_{\text{retry}}$
 1: Initialize:
        $\bar{\lambda}^i = 0$, $u^i_{t-1} = 0$, $\langle \bar{sr}^1, \ldots, \bar{sr}^M \rangle = \mathbf{0}$
 2: for $t = 1, 2, \ldots$ do
 3:     Generate packets
 4:     Append packets to buffer until $L_{\text{buffer}}$ packets are stored
 5:     if $t$ mod $\tau = 0$ then
 6:         Select resource unit $u^i_t = \text{argmax}_{u^i}\ Q_i(z^i, u^i; \theta^i)$
 7:     end if
 8:     Back-off or transmit the head packet in the buffer on resource
        unit $u^i_t$ (see Section II-C)
 9:     Receive acknowledgement packet from BS
10:     Adjust back-off parameter $cw$ or drop packet (see Section
        II-C)
11:     Update local observation $z^i$ includes $\bar{\lambda}^i$, $\langle \bar{sr}^1, \ldots, \bar{sr}^M \rangle$
        using Formula 1
12:     Update the running mean and variance of $z^i$ at the training
        phase, and normalize $z^i$ in all phases
13: end for
```

In every timeslot, each device $i \in \{1, \ldots, N'\}$ chooses an action $u^i \in U$, which is a resource unit. Here, U is a group of resources, which is granted a group of devices. The actions of all devices in that group constitute the joint action $\mathbf{u} = (u^1, \ldots, u^{N'}) \in \mathbf{U}$. If the joint action is $\mathbf{u}$ under the state s, the next state of the system s' can be obtained according to the state transition function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \to \mathbf{S}$. For this Dec-POMDP, we assume it is of a model-free structure. During the training phase, the network simulator can generate the transition function P. Given the current network state s and the joint-action $\mathbf{u}$, the simulator computes the next state s' according to the traffic model, binary exponential back-off rule, and buffer rule, which is described in Section II. We discuss the remaining elements of the Dec-POMDP formulation in the following subsections.

A. Measurement of local observation

In Dec-POMDP, partially observable means that each of n agents cannot obtain the true state s because s contains a network-wide information. However, devices can extract their local observation $z \in Z$, which can be easily gathered on their own. The local observation is the input for devices to determine the next action without communicating to the BS. We design the local observation as follows:

1) the average packet arrival rate $\bar{\lambda}^i$
2) the previous action u^i_{t-1}
3) the list of average success transmission rate per resource $\langle \bar{sr}^1, \ldots, \bar{sr}^M \rangle$.

The first two elements represent the internal state of each device, whereas the remaining elements capture partial information about the network's traffic intensity at each resource from the perspective of that device.

Each device selects an action based on a stochastic policy $\pi^i(u)$ (Line 5-7, Algorithm 1). In particular, the policy is maximizing the Q-value $\text{argmax}_{u^i}\ q_i = Q_i(z^i, u^i; \theta^i)$, where θ^i is the deep neural networks (DNN)'s parameter. Previous works adopted a sequence of historical local observation as the input for each agent and LSTM as the network architecture [5], [7], [8]. This approach demanded a large memory footprint and lengthy computation, but mMTC devices have limited computational capability. Thus, a compact local observation and small neural networks architecture is needed. The local observation for agents in our proposed system is a vector of $M+2$ elements. Also, a fully connected DNN with one hidden layer is the neural networks architecture. Besides, to minimize the memory footprint on the mMTC devices, the historical average is captured via an incremental implementation [12, p.31]. For example, the update rule to estimate the average packet arrival rate is:

$$\bar{\lambda}^i_{t+1} \leftarrow \bar{\lambda}^i_t + \alpha(x_t - \bar{\lambda}^i_t) \tag{1}$$

where x_t is the number of packet generated within the t^{th} timeslot, and α is the step size of the update. A similar update rule is applied for estimating the success rates.

The learning process could be hindered if the scale of different inputs to the DNN are not the same. Also, agents do not know the exact scale of the average packet arrival rate as well as the average success transmission rate. Thus, we track the running mean and variance of the local observations using the data generated during the training phase of the system [13]. This technique allows the mean and variance to be estimated as the observation arrives one at a time, while devices do not need to keep the observation for a second pass. Devices learn the means and variances during the training phase. Then, the final means and variances of the observations are kept constant to normalize the observation in the testing and deployment phase (Line 12, Algorithm 1).

B. Global observation and reward

The global observation is the concatenation of local observations $\mathbf{z} = \langle z_1, \ldots, z_{N'} \rangle$. This global observation is the input for the Hypernetwork of QMIX, which generates the parameter for the mixer network. The mixer network combines the values given by individual value functions $\mathbf{q} = \langle q_1, \ldots, q_{N'} \rangle$ into a single value $\hat{q}_{\text{tot}}$, which estimate the joint-action's value. In the training phase, the gradient is backpropagated from the mixer network to individual DNN value functions. This mechanism allows each device to learn the association between its local observation and the global observation, thereby leading to higher performance than independent DQN.

Let $r(s, \mathbf{u}) : S \times \mathbf{U} \rightarrow \mathbb{R}$ be the joint-action reward function. Consider cluster of devices $\mathcal{N}^{'}$ at timeslot t, the set of devices that attempt to transmit is $\mathcal{N}^{'}_{\text{transmit}}(t)$. The set of successfully transmitted devices is $\mathcal{N}^{'}_{\text{success}}(t)$. Then, the joint-action reward at every scheduling interval τ is defined as:

$$r(t) = \frac{1}{N'} \sum_{i=1}^{i=N'} \frac{\sum_{t'=t}^{t'=t+\tau} 1\{i \in \mathcal{N}^{'}_{\text{success}}(t)\}}{\sum_{t'=t}^{t'=t+\tau} 1\{i \in \mathcal{N}^{'}_{\text{transmit}}(t)\}} \tag{2}$$

Equation (2) presents the average success transmission rate overall $N^{'}$ devices in a scheduling interval τ. If the reward is higher, the average access delay is lower because a high success rate means that devices do not have to perform am excessive amount of back-off operations. This reward function is similar to previous work on sporadic traffic [9]. The reward is chosen to be the average success rate over a longer time horizon such that the uncertainty of the sporadic traffic arrival is mitigated.

C. Training with dynamic traffic

TinyQMIX agents can be trained to select the best wireless resource under various traffic conditions according to Algorithm 2. First, the DNN for every device and the mixer network are initialized at random. The average arrival rate is changed every interval ΔT (Line 3-5, Algorithm 2). We train the TinyQMIX agents such that they can cooperatively select the resource units in an uncoordinated manner. Particularly, the TinyQMIX parameter is optimized to estimate the correct joint-action Q-value with different sporadic traffic distributions (Line 13-16, Algorithm 2).

Algorithm 2 An episode of offline centralized training

```
1:  $t \leftarrow 0$
2:  while $t < T_{episode}$ do
3:      if $t$ mod $\Delta T = 0$ then
4:          Redraw $\lambda(t)$
5:      end if
6:      for step in $1, \ldots, T_{\text{optimization interval}}$ do
7:          Run all agents (Algorithm 1)
8:          Collect local data $\mathbf{z}_t$, $\mathbf{z}_{t+1}$, $\mathbf{u}_t$
9:          Collect global reward $r(t)$
10:         Save $\langle \mathbf{z}_t, \mathbf{z}_{t+1}, \mathbf{u}_t, r(t) \rangle$ to replay memory
11:     end for
12:     Sample minibatch from replay memory
13:     Compute individual's Q-values
            $\mathbf{q} = \langle Q_i(z^i, u^i; \theta^i), \forall i \in \{1, \ldots, N'\} \rangle$
14:     Compute estimated total Q-value
            $\hat{q}_{tot} = Q_{tot}(\mathbf{z}, \mathbf{u}; \theta^{\text{mixer}})(\mathbf{q})$
15:     Compute target total Q-value
            $q_{tot} = r + \gamma \max_{\mathbf{u'}} Q_{tot}(\mathbf{z}, \mathbf{u'}; \theta^{\text{mixer}})$
16:     Compute the mean square error between $\hat{q}_{tot}$ and $q_{tot}$.
        Compute the gradient using the error and update the network
        parameters using stochastic gradient ascend
17:     $t \leftarrow t + 1$
18: end while
```

In the testing phase, we generate and store traffic traces. A trace is a matrix containing the number of packets generated by each device in every timeslot. Different methods are tested on the same trace to compare their performance. When being tested on the trace, the trained TinyQMIX models do not require any further fine-tuning or adaptation. The trained model can immediately perform well on newly unseen traffic distribution in the testing traffic traces because the DNN agents can generalize and give an accurate assessment of the action's value based on the vast number of trained patterns.

IV. Numerical Simulation

In this section, we discuss the simulation scenario, briefly introduce the practical baselines, and finally provide an empirical comparison.

A. Simulation scenario

We tested different channel access policies under three levels of traffic dynamic: $\Delta T \in \{10\text{s}, 60\text{s}, \infty\}$. The traffic of the set of high active devices $\mathcal{N}_h$ has Poisson distribution with average arrival rate $\lambda_h = 0.1$ (packet per slot), whereas that of the low active devices $\mathcal{N}_l$ also has Poisson distribution with $\lambda_l = 0.00833$ [9]. The device type is randomly reassigned such that the probability of high active device is $1/5$, and the probability of low active device is $4/5$.

The total length of the traffic traces for testing is 1 hour. We consider the 0.5 ms timeslot, then the testing time is equal to 7.2 million timeslots. Let the scheduling interval be 25ms (50 timeslots), then there are a total of $144k$ scheduling intervals. Also, the MAC's parameters L_{buffer}, L_{retry}, and CW_{max} are all equal to 16. These parameters were chosen such that the probability of packet drop is negligible. Thus, different policies are only compared in terms of their access delay.

Then, we tested the system with different cluster sizes. The number of resource units per cluster are $M \in \{2, 4, 8, 16\}$, and the number of devices per cluster are $N' \in \{12, 24, 48, 96\}$, respectively. The system overloading is $\phi = 6$. Similar to [9], the ratio between high and low active devices is $\frac{|\mathcal{N}_h|}{|\mathcal{N}_l|} = \frac{1}{4}$.The detailed hyperparameters for training TinyQMIX are presented in Table I.

TABLE I: Hyperparameters for training TinyQMIX

Hyper parameters	Value
Number of training episodes	1000
Episode length	100(s)
Optimization interval	32
Learning rate	1e-4
Batch size	1024
Replay memory size	10000
Discounted factor γ	0
Exploration start ϵ_{start}	0.9
Exploration end ϵ_{end}	0.05
Number of agents	{12, 24, 48, 96}
Observation update's step size α	0.001
Number of hidden units for agents' DNN	{8, 8, 16, 32}
Number of hidden units for mixer's DNN	{64, 128, 256, 512}

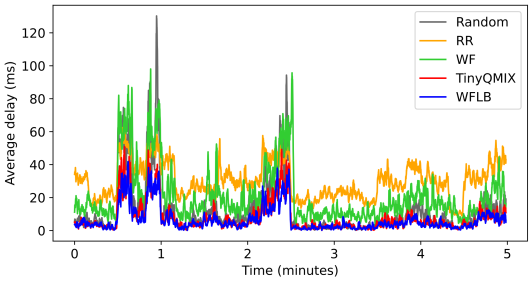


Fig. 2: Moving average of delay in the first 5 minutes of the testing trace, with $\Delta T = 10$s and $N' = 12$.

B. Baselines

We compare TinyQMIX policy with widely-used distributed resource allocation policies. First, random is a policy that each device selects the action randomly. Second, Round Robin (RR) is a policy in which each device takes turn for transmitting sequentially in each available resource unit and sequentially over time. DQL which has been proposed in [9] is also taken into account. We also compare our proposal with recently proposed QMIX policy using LSTM as agent's network architecture [7], [8], denotes LSTMQMIX. Similar to what has been done in [8], we kept a sequence of 10 local observations as the input for the LSTM agents. Then, we trained the LSTMQMIX policy using Algorithm 2.

Water Filling (WF) is a heuristic and centralized policy, in which, the BS can estimate the average arrival rate to allocate a balanced amount of traffic on different resource units. However, in WF, devices must send the CSI to BS, then BS must take a proportion of timeslots to send the resource assignment to devices. Water Filling Lower Bound (WFLB) is an unrealistic version of WF. In WFLB, we assume that there is no signaling overhead and the perfect arrival rate is known beforehand. Because of its unrealistic assumptions, WFLB has the best performance and it will serve as an empirical performance bound. [1]

C. System performance

We compare the performance of the proposed method in different simulation scenarios.

1) Compare delay over time when the traffic is highly dynamic: Figure 2 compares the delay of random, RR, WF and TinyQMIX policy under the scenario that traffic condition changes every 10 seconds. The delay of WF was among the highest because $6/50$ timeslots are reserved for downlink resource assignment, while these timeslots can be used for uplink data in other distributed methods. The performance of RR was far from the lower-bound performance of WFLB. Although RR does not have any signaling overhead like WF, its average delay still fluctuated around 40 ms, because the arrived packets need to wait for a long time until the scheduled timeslot.

TinyQMIX consistently outperformed other policies. Its delay approached the lower bound method WFLB throughout 5 minutes testing trace, in either low traffic intensity or high traffic intensity situation.

2) Compare delay over cluster size: Figure 3a presents the average delay of different policies when the number of devices per cluster increases. TinyQMIX consistently outperformed the other policies. As the cluster scaling up, DQL and LSTMQMIX became worse, while WF improved slightly but all of them were not outperform TinyQMIX. It shows that TinyQMIX can consistently handle the task of distributed resource unit selection better than the baselines on all tested network scales. When the size of the cluster increases, the delay of WFLB became smaller because there is higher flexibility for selecting resource units. However, the joint-action space of multiple agents also became exponentially larger, which makes training the TinyQMIX harder. The results suggest that the best cluster size is 24, which TinyQMIX can produce the lowest delay, in comparison with other cluster sizes.

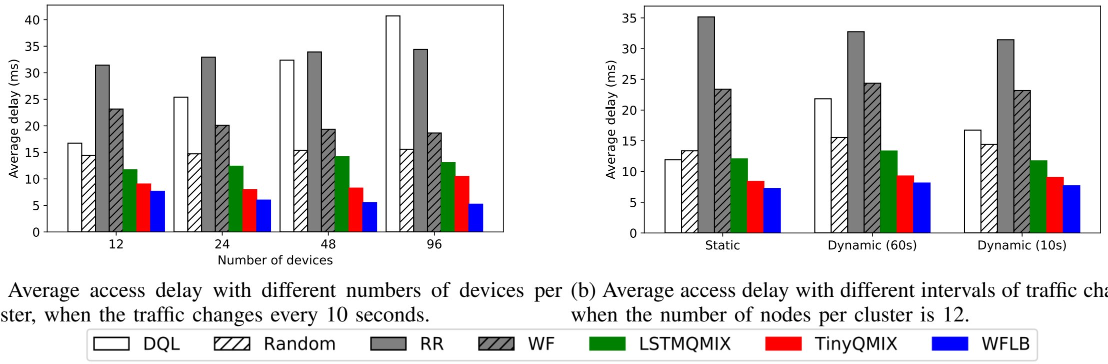
3) Compare delay over different traffic dynamic scenarios: Figure 3b compares different policies under three traffic dynamic scenarios. We are able to reproduce the result from [9], that is DQL is better than random and RR policies under the static traffic trace. However, the average delay of DQL was higher than a random policy at a higher traffic dynamic, which indicates that DQL cannot handle dynamic traffic as we hypothesized. Besides, there is no significant change in the delay produced by RR, random, or WF policies in all three testing traces.

LSTMQMIX and TinyQMIX were trained on the scenario in which $\Delta T = 10$s and tested on slower traffic changes. LSTMQMIX only performed well on the trained scenario, while TinyQMIX performed well in all cases. This indicates that TinyQMIX generalized the trained environment better than LSTMQMIX. In all traces, the gap between the delay of TinyQMIX and WFLB was always the smallest. These results suggest that the proposed TinyQMIX policy is the most suitable policy for tackling both static and highly dynamic traffic.

D. Model complexity

Here, we compare the model complexity of different fully distributed policies. Note that DQL or WF requires regular information exchange between BS and devices, thereby comparing the model complexity of fully uncoordinated policy such as TinyQMIX with DQL or WF is unfair. We compare fully distributed policies such as Random, TinyQMIX, and LSTMQMIX in terms of their FLoating-point Oerations Per Second (FLOPS) needed to compute the local observation and perform model inference. Random has the smallest computational requirement at only 40 FLOPS for selecting 40 random actions per second. On the other hand, TinyQMIX consumes 3000 FLOPS when $N' = 12$, 4520 FLOPS when

[1] The full implementation of our experiment can be found at https://github.com/lethanh-96/tinyqmix-mtc



(a) Average access delay with different numbers of devices per cluster, when the traffic changes every 10 seconds.

(b) Average access delay with different intervals of traffic changes, when the number of nodes per cluster is 12.

Fig. 3: Average access delay under different network sizes and traffic changing rate

$N' = 24$, and just below 50k FLOPS when $N' = 96$. LSTMQMIX is the most demanding agent, which requires at least 52k FLOPS for the smallest subgroup size of 12. Note that, common general purpose microprocessor such as ARM Cortex-M has the maximum computational capacity of 1.6 GFLOPS, thereby it can clearly support TinyQMIX with the best subgroup size $N' = 24$. In short, TinyQMIX can facilitate a smaller delay than a simple method like Random, and induces significantly less computational overhead than the recently proposed MADRL policy such as LSTMQMIX.

V. Discussion & Concluding Remarks

To sum up, our work proposed TinyQMIX, a lightweight cooperative multi-agent reinforcement learning policy that enables distributed and autonomous mMTC network. The findings of this study support the idea that a proper distributed resource allocation method can outperform a centralized method, due to the cost of exchanging information with a centralized controller. Besides, our results support the hypothesis that reinforcement learning, which is learned on various patterns, can generalize and adapt to changes.

References

[1] M. B. Shahab, R. Abbas, M. Shirvanimoghaddam, and S. J. Johnson, “Grant-free non-orthogonal multiple access for iot: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1805–1838, 2020.

[2] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, and J. M. Lopez-Soler, “A survey on 5g usage scenarios and traffic models,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 905–929, 2020.

[3] X. Chen, D. W. K. Ng, W. Yu, E. G. Larsson, N. Al-Dhahir, and R. Schober, “Massive access for 5g and beyond,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 3, pp. 615–637, 2020.

[4] K. Au, L. Zhang, H. Nikopour, *et al.*, “Uplink contention based scma for 5g radio access,” in *2014 IEEE Globecom workshops (GC wkshps)*, IEEE, 2014, pp. 900–905.

[5] J. Zhang, X. Tao, H. Wu, N. Zhang, and X. Zhang, “Deep reinforcement learning for throughput improvement of the uplink grant-free noma system,” *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6369–6379, 2020.

[6] T. Rashid, M. Samvelyan, C. Schroeder, G. Farquhar, J. Foerster, and S. Whiteson, “Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning,” in *International Conference on Machine Learning*, PMLR, 2018, pp. 4295–4304.

[7] R. Huang, V. W. Wong, and R. Schober, “Throughput optimization for grant-free multiple access with multi-agent deep reinforcement learning,” *IEEE Transactions on Wireless Communications*, vol. 20, no. 1, pp. 228–242, 2020.

[8] Z. Guo, Z. Chen, P. Liu, J. Luo, X. Yang, and X. Sun, “Multi-agent reinforcement learning based distributed channel access for next generation wireless networks,” *IEEE Journal on Selected Areas in Communications*, 2022.

[9] J. Liu, Z. Shi, S. Zhang, and N. Kato, “Distributed q-learning aided uplink grant-free noma for massive machine-type communications,” *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2029–2041, 2021.

[10] H. Jiang, Q. Cui, Y. Gu, X. Qin, X. Zhang, and X. Tao, “Distributed layered grant-free non-orthogonal multiple access for massive mtc,” in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, IEEE, 2018, pp. 1–7.

[11] F. A. Oliehoek and C. Amato, *A concise introduction to decentralized POMDPs*. Springer, 2016.

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

[13] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms (3rd Edition)*. Addison-Wesley Professional, 1997.