Decentralized Configuration of TSCH-Based IoT Networks for Distinctive QoS: A Deep Reinforcement Learning Approach

Hamideh Hajizadeh, Majid Nabi, *Member, IEEE*, Kees Goossens, *Member, IEEE*

***Abstract*—The IEEE 802.15.4 Time-Slotted Channel Hopping (TSCH) is widely used as a reliable, low-power, and low-cost communication technology for many industrial Internet-of-Things (IoT) networks. In many applications, Quality-of-Service (QoS) requirements are different for heterogeneous nodes, necessitating non-equal parameter settings per node. This results in a very large configuration space making space exploration complex and time-consuming. Moreover, network state and QoS requirements may change over time. Thus, run-time configuration mechanisms are needed for making decisions about proper node settings to consistently satisfy diverse and dynamic QoS requirements. In this paper, we propose a run-time decentralized self-optimization framework based on Deep Reinforcement Learning (DRL) for parameter configuration of a multi-hop TSCH network. DRL adopts neural networks as approximate functions to speed up the process of converging to QoS-satisfying configurations. Simulation results show that our proposed framework enables the network to use the right configuration settings according to the diverse QoS demands of different nodes. Moreover, it is shown that the convergence time of the learning framework is in the order of a few minutes which is acceptable for many IoT applications.**

***Index Terms*—IoT, IEEE 802.15.4 TSCH, WSN, DRL, QoS**

I. Introduction

THE IEEE 802.15.4 [1] standard specifies Physical (PHY) and Medium Access Control (MAC) layers of low-rate and low-power Wireless Sensor Networks (WSNs). Time-Slotted Channel Hopping (TSCH) as a MAC operational mode of this standard is introduced to provide predictable and reliable communication mainly targeting industrial applications. To achieve these objectives, TSCH uses time-slotted access, multi-channel communications, and channel hopping mechanisms. The TSCH protocol supports two medium access methods using dedicated and shared timeslots. A dedicated timeslot is exclusively assigned to a link in a neighborhood for collision-free communication. Clear Channel Assessment (CCA) may be used in dedicated timeslots to combat external interference by preventing transmission in the dedicated timeslot. On the other hand, a shared timeslot is allocated for communications of multiple links, which can lead to collisions. A CSMA/CA mechanism is used in the MAC layer to handle contentions and repeated collisions in the shared timeslots.

TSCH CSMA/CA is fundamentally different than that of the other MAC modes of the IEEE 802.15.4 standard (e.g., non-beacon enabled mode) as it does not perform carrier sensing before activating the back-off algorithm. Instead, it follows an ALOHA-based transmission in the first shared timeslot and performs back-off for a certain number of shared timeslots in case of a collision. Dedicated access has had the main focus of being used in many industrial applications since it provides more deterministic channel access and thus more predictable performance. However, shared access has the potential to be employed in certain applications due to its simplicity as it does not need a complicated scheduler like what dedicated timeslots require. Moreover, in sparse networks or applications with low data traffic, shared access can provide better end-to-end latency. It is while the shared timeslots can be used in combination with dedicated timeslots to better support a wide range of industrial applications. In this paper, we focus on shared timeslots and the configuration of the TSCH CSMA/CA mechanism since its parameters have a great impact on network performance.

There are two requirements for an effective optimization mechanism for setting the parameters of the TSCH CSMA/CA mechanism. First, our simulations show that a homogeneous setting of parameters in a (part of a) network is far from optimum. It is because of the inherent heterogeneity we typically have in the network in terms of sensing characteristics, node's position in the routing structure, and diverse application Quality-of-Service (QoS) requirements. Second, network dynamics require run-time adaptation of the MAC parameters of individual nodes. Recent studies such as [2]–[4] present centralized optimization techniques using performance models to set MAC parameters and tune them at run-time. However, in many IoT networks, it is not possible to derive a fast and accurate enough performance model. Moreover, even if such a performance model is available, a centralized approach is not agile enough to sense the changes, compute new settings, and distribute them to the nodes in a large network. We consider

This work was supported by the SCOTT European project (www.scott-project.eu that has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No737422.

The authors are with the Department of Electrical Engineering, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands. Majid Nabi is also with the Department of Electrical and Computer Engineering, Isfahan University of Technology, Isfahan 84156-83111, Iran (email: {h.hajizadeh, m.nabi, k.g.w.goossens}@tue.nl).

multi-hop TSCH networks with a tree-based routing structure (e.g., RPL [5]) that exploit shared timeslots for data or control packet exchange. The contributions of this paper are as follows.

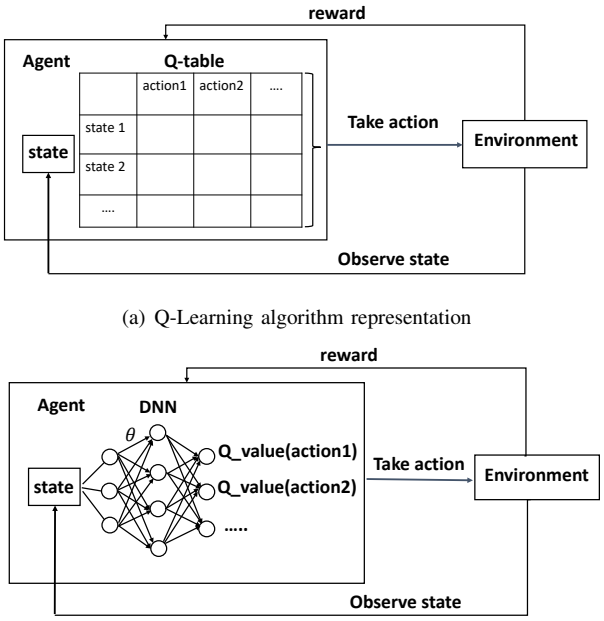
1) A decentralized optimization framework is designed for setting the CSMA/CA parameters of each node individually based on its properties to support their diversified characteristics and QoS requirements. In a tree-based routing structure, the parent node at each level receives characteristics of its nodes and sends back the proper settings for their channel access.
2) As the optimization engine, we map the problem on a model-free Deep Reinforcement Learning (DRL) technique. DRL is a machine learning technique that combines deep learning and Reinforcement Learning (RL) to solve complex decision-making tasks. The main reason for this approach is the lack of accurate models for extracting the impact of the CSMA/CA parameters on high-level QoS metrics in a multi-hop dynamic network. The state-action space and the reward function are defined to improve the convergence time of the technique.
3) Extensive simulations are performed to study the effectiveness of the developed scheme. The results show that the framework can meet diverse QoS demands and reconfigure itself quickly when the network state changes to follow time-varying conditions.

The remainder of the paper is organized as follows. Section II gives an overview of the TSCH CSMA/CA standard and the Reinforcement Learning (RL) methods that are used in this work. Section III reviews the related work on MAC parameters setting in WSNs. The system model and problem statement are presented in Section IV. Section V describes our learning-based method for run-time self-configuration of a TSCH network. The evaluation of the performance of the proposed algorithm is presented and discussed in Section VI. Section VII discusses the constraints of deploying our proposed algorithms in real-world IoT devices. Section VIII concludes.

II. Background

A. IEEE 802.15.4 TSCH CSMA/CA

TSCH CSMA/CA back-off mechanism is performed by the nodes transmitting in the shared timeslots to avoid repeated collisions. It is different than the original CSMA/CA technique in IEEE 802.15.4. A node performing the TSCH CSMA/CA transmits its data packet in the first shared timeslot (no carrier sensing with clear channel detection). In case of not receiving an Acknowledge (ACK) packet, a back-off procedure is activated and the back-off parameters are initialized, i.e., the Back-off Exponent (*BE*) is set to its minimum value (BE^{min}), and the number of retransmission performed by the node is set to zero ($RT = 0$). Then, a random number w is picked in the range $[0, 2^{BE} - 1]$. The node waits for w shared timeslots and then transmits its data packet unless it reaches its dedicated timeslot, in which it can exclusively transmit its data packet. If the (re)transmission occurs in a shared timeslot with successfully received ACK, the algorithm terminates. If it could not receive ACK, *RT* is increased by one to count the number of transmission failures, and *BE* is increased by one up to its maximum value (BE^{max}) to double the size of the contention window in order to reduce the chance of collisions. If the number of retransmissions exceeds its maximum allowed number (*maxR*), the data packet is dropped. When retransmission of the data packet succeeds in a dedicated link, *BE* does not change if the transmission queue is still not empty afterward. Otherwise, it resets to BE^{min}.



(a) Q-Learning algorithm representation

(b) Deep Q-Learning algorithm representation

Fig. 1. General structure of Q-Learning and Deep Q-Learning algorithms

BE^{min}, BE^{max}, and *maxR* are the MAC parameters of a TSCH node that should be configured. Each configuration parameter has a different effect on the performance of the network in terms of reliability, latency, and energy consumption. In this paper, we present a self-optimization technique to configure TSCH nodes according to their QoS demands.

B. Reinforcement Learning

This section reviews the RL concepts and techniques [6]. In an RL framework, a decision-making agent interacts with the environment in consecutive time steps. At time step t, the agent observes the environment state s_t and executes an action a_t selected from the set of all possible actions A according to a policy π, which is a mapping from states to actions. When the agent executes action a_t, it receives a reward r_{t+1}, and the environment enters the next state s_{t+1}. There are several RL techniques such as policy optimization methods, policy gradient methods, and value function-based methods (e.g., Q-learning methods) [7]. In this paper, we use the Q-learning technique [8] and its extension, Deep Q-learning [9] for our system model.

*1) **Q-Learning**:* Q-Learning (QL) is a model-free RL algorithm that aims to find the best action to execute given the current state without knowing the policy. In this technique, given a series of experiences $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$, the cumulative discounted reward at t is calculated by $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where $\gamma \in (0, 1]$ is the discounted factor. For a specific policy π, Q-value of a state-action pair (s, a) is the expected R_t and is calculated as $Q^{\pi}(s, a) = Expectation[R_t | s_t = s, a_t = a, \pi]$. QL intends to find the optimal policy, $Q^*(s, a)$, among all policies so that $Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a)$. Accordingly, the agent at time t takes action a_t and moves from state s_t to the state s_{t+1}. It gets a reward r_{t+1}, and then updates the Q function as follows.

$$Q_{new}(s_t, a_t) \longleftarrow (1 - \alpha) \times Q_{old}(s_t, a_t) + \alpha \big[r_{t+1} + \gamma \max_{a} Q_{old}(s_{t+1}, a) \big], \tag{1}$$

where $\alpha \in (0, 1]$ is the learning rate parameter. QL is represented as a tabular format called Q-table, in which Q-values for state-action pairs are stored in the table shown in Fig. 1(a).

There is a well-known method in QL, called ϵ-greedy algorithm, to choose an action at time t . In this algorithm, the agent selects a random action from all possible actions with probability ϵ called exploration, and the action $a_t = \arg\max_a Q(s_t, a)$ with probability $1 - \epsilon$ called exploitation. Enough exploration, especially in the early stages, avoids the agent to get stuck to exploit a locally optimal policy. Hence, the selection of the proper value for ϵ is important. It should be set to one at early stages to explore a wide range of actions and then decreased with a proper approach to exploit more actions after a reasonable amount of learning steps.

*2) **Deep Q-Learning**:* Watkins et all have shown in [8] that, in a stationary Markovian environment, the Q function converges to the optimum Q-values with probability 1 if all actions are repeatedly sampled in all states and the action-values are represented discretely. However, many real-world problems have huge state and/or action spaces, in which convergence to the optimal policy takes a considerably long time. If the environment changes in the meantime, the QL algorithm never converges to $Q^*(s, a)$. In these scenarios, the learning information from one experience set e_t is generalized to other similar new situations. To do so, function approximation methods are mainly used to approximate the Q-values [6]. In [9], the authors proposed an algorithm called Deep Q-Learning (DQL), in which a Deep Neural Network (DNN) model is used to approximate the Q function for each action value, called Q Neural Network (QNN).

Fig. 1(b) shows DQL algorithm wherein the states are applied as inputs to the neural network. It outputs the estimated Q-values for all possible actions, $\{Q(s, a; \theta) | a \in A\}$, where θ is the set of weights of the edges in the QNN. DQL employs the same ϵ-greedy algorithm for action selection as in QL where in the exploitation case, $a_t = \arg\max_a Q(s_t, a)$ is replaced by $a_t = \arg\max_a Q(s_t, a; \theta)$. To estimate the Q-values, $Q(s_t, a_t; \theta)$, the QNN should be trained and its parameters, θ values, should be updated by minimizing the loss function [9]. In particular, the loss function at time step t when the experience e_t is explored will be calculated as,

$$Loss(\theta, e_t) = \big[Q(s_t, a_t; \theta) - (r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a, \theta^{next})) \big]^2, \tag{2}$$

where $Q(s_t, a_t; \theta)$ is the predicted Q-value approximated by the QNN and $r_{t+1} + \gamma \max_a Q(s_{t+1}, a, \theta^{next})$ is the target Q-value of the QNN based on the experience $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$. Note that, QNN's training in DQL differs from the training of traditional neural networks in supervised learning, where the weights of the network are tuned offline. The QNN's weights are updated using the last experience in an online manner.

III. Related Work

This section provides an overview of the most relevant efforts in the literature to configuration and run-time adaptation of the MAC layer parameters in WSNs. Several works have focused on centralized model-based MAC parameters' configuration. [3] and [4] introduce platforms including an optimization engine as the main core for the configuration of MAC parameters at run-time to cope with network dynamics. The optimization engine of these platforms requires a performance model to quickly estimate network performance for a given configuration. Being centralized, such platforms demand a long time to detect changes in the network, find optimum settings for the new state of the network, and upload them into the wireless nodes, especially in large-scale multi-hop networks. Our earlier work [2] tries to speed up the optimization engine for such centralized configuration schemes using model-based approximated Pareto set to tune CSMA/CA MAC parameters of a TSCH network. However, homogeneous networks are considered in these frameworks in which all nodes take equal CSMA/CA parameter settings. We show, in this paper, that such equal settings do not work for networks that have nodes with different packet arrival rates and/or QoS demands. Moreover, such approaches require performance models to estimate the performance of the network which is not available for many real-world IoT networks, especially for non-homogeneous networks.

RL is introduced as a model-free decision-making technique that can be used for the intelligent configuration of wireless networks. Through trial and error in an interactive environment, RL agents can autonomously learn implicit knowledge of network dynamics from raw high-dimensional observations. [10] gives a survey on RL techniques used for WSNs in PHY, MAC, and network layers. Since our paper focuses on learning-based optimization and self-configuration techniques in the MAC layer, we limit our review to the same area.

[11] is the most relevant work. It first analyzes the performance of the dynamic back-off parameter (*BE*) setting in the base CSMA/CA of IEEE 802.15.4 and verifies the essence of dynamic configuration according to the traffic rate. A QL

algorithm is proposed to configure the control and networking subsystems. Although it considers equal settings (*BE* value) for all nodes and ignores heterogeneity making state-action space with a reasonable size to explore, it still suffers from a low convergence speed of the learning algorithm. With a large number of state-action pairs, the convergence speed of QL will be an issue since each state-action value must be explored at least once. To tackle this problem, [11] suggests two priority-aware policies to improve the convergence speed. The first policy is to reduce the number of states from a complete set to a limited set by grouping some states together and representing them as one state. The second policy is to assign a limited number of actions per state according to the knowledge of the traffic in the network, which needs prior experiments. However, for a very large number of state-action spaces with dynamic environments, the problem still remains in place. In our work, we consider the CSMA/CA algorithm of the TSCH mode, which is fundamentally different than the base mechanism. Moreover, we consider individually setting of nodes to support heterogeneous specifications and QoS requirements of various nodes. It leads to a substantially larger state-action space for which the QL-based solution proposed in [11] will not be a feasible approach.

Recent advances in deep learning open a new area for RL called DRL. Deep Q-Network (DQN) [10] is a representative of DRL, which embraces the advantage of DNN in approximating the value functions, and hence speeding up the learning process and improving the RL performance for environments with large states/actions. [12] presents a survey on the applications of DRL in QoS provisioning at the MAC layer including medium access and data rate control, resource sharing, and scheduling. They highlighted the main advantages of DRL-based methods compared to traditional methods for various problems.

[13]–[15] are some efforts utilizing the DRL mechanism for MAC parameter configuration in WSNs. In [13], the authors present a DQN-based mechanism on a transmission and back-off approach that consists of two sub-networks. The first sub-network is a DQN for the configuration of the time scheduler in order to transmit more low-priority packets while not affecting the performance of high-priority packets. The second sub-network which is also a DQN derives optimal back-off length by using the action of the first sub-network and channel occupancy. Simulation results prove the effectiveness of the proposed DQN-based approach under different traffic loads. [14] utilizes DRL to adjust the length of the initial back-off window during the back-off process to increase the uplink throughput and reduce the energy outage probability. It is shown that this method improves throughput compared to traditional techniques. In [13] and [14], homogeneous nodes are considered regardless of the fact that in real-world applications, WSNs may need to support heterogeneous nodes. [15] presents an inner-state-driven random access framework for wireless networks with heterogeneous nodes having diversified QoS requirements. They use policy optimization methods for the configuration of the time scheduler which schedules the transmission opportunities among nodes by mapping the inner states of each node to a transmission probability. Neural networks are adopted as approximation function mapping nodes' inner states to actions. Experimental results show that, in a simplified case with 3 heterogeneous nodes, the QoS of all nodes can be improved simultaneously by their framework compared to the conventional approaches which use equal transmission probability for all nodes.

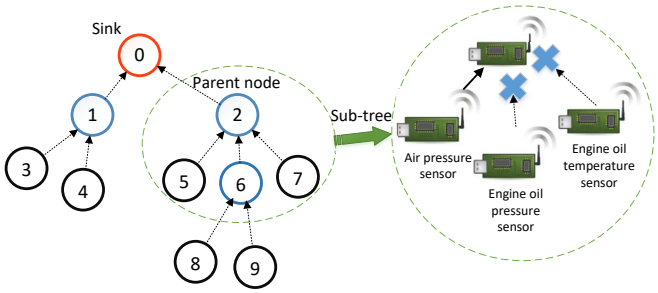


Fig. 2. An illustration of the system model showing diversified QoS requirements in a TSCH network.

This paper is the first attempt to provide a model-free decentralized run-time self-configuration platform for TSCH CSMA/CA supporting heterogeneous nodes with diverse and dynamic QoS requirements running in a tree-based multi-hop network.

IV. System model and Problem statement

A. System model

A converge-cast WSN is considered in which each node intends to send its data packets to a sink node. Each node runs the TSCH standard as the MAC layer and a single-parent tree routing mechanism such as RPL [5] as its multi-hop routing protocol. Fig. 2 shows an example of such a network as a motivating control application deployed in a vehicle, wherein different sensors have to monitor environmental parameters and continuously transmit their data to the sink node through a routing path. According to the sensor's type, the network must support differentiated QoS. Data packets arrive at the MAC layer of nodes from their own sensor or from the nodes in their sub-tree; such packets are aggregated and then transmitted to the parent node in each sub-tree. It is assumed that all nodes in the same sub-tree use only shared timeslots for transmission of their data to their parent, like the mechanism used in the widely-used Orchestra scheduler [16] in the receiver-based mode. The configuration of each node in a sub-tree affects the performance of other nodes in the same sub-tree since they compete with each other in the shared links to deliver their data to their parent.

The packet arrival rates of the nodes and their patterns (periodic or event-based) may be different and time-variant. A very common packet management scheme is considered, in which older packets are dropped whenever a newer version of the packet arrives. To avoid high signaling overhead and have an acceptable response time, the decentralized configuration

of TSCH parameters is the problem of interest. The parent node in each sub-tree receives control packets from its children about their data traffic specification and the QoS requirements. It then extracts appropriate settings of the TSCH CSMA/CA parameters of all the nodes in the sub-tree and broadcasts them to its children. This makes the problem decentralized at the network level and centralized at the sub-tree level. While a parent node makes decisions about the settings of its children based on the specifications it locally receives from them, such information has been gradually made in the whole hierarchy of the tree-based routing structure. The specifications and requirements are developed by the children nodes based on their own sub-tree. The nodes aggregate their own specifications with those of their sub-tree and then send them to their parents.

By the aforementioned configuration strategy, the problem is narrowed down to the configuration of the nodes within a sub-tree by the parent node. Consider a sub-tree with N nodes connected to a parent node indicated by $n_1, n_2, ..., n_N$. Each node may have a number of configuration parameters and the sub-tree as a whole has p configuration parameters. P_k indicates the set of all values that k-th parameter can get ($1 \leq k \leq p$). Also, a number of performance metrics may be defined for each node independently; QoS_i indicates a QoS metric for node n_i. Configuration space of such sub-tree is then the set of all possible configurations, which is the Cartesian product of finite configuration parameters, as $C = P_1 \times P_2 \times ... \times P_p$. $c = (c_1, c_2, ..., c_p)$ is an element of such a configuration space which leads to $QoS_i(c)$ for $1 \leq i \leq N$ and any QoS defined for node n_i. Let Q_{obj} and Q_{cnt} be the subsets of the set of all QoS metrics considered as objectives and constraints, respectively. Note that QoS metrics of node n_i are affected by the configuration of all nodes in the sub-tree, not only by its own settings. The optimization problem is stated in a generic form as follows.

$$\begin{array}{lll} optimise & QoS(c) & \forall\ QoS \in Q_{obj} \\ c \in C & & \\ s.t. & QoS_j(c) \succeq QoS_j^{req} & \forall\ QoS_j \in Q_{cnt} \end{array}, \quad (3)$$

where the QoS metrics in Q_{obj} have to be optimized as objectives, and for every QoS metrics QoS_j in Q_{cnt}, its corresponding constraint, QoS_j^{req}, needs to be met. Notation $\succeq$ represents better performance, which may be a lower or higher value depending on the nature of the metric.

Since the network may be heterogeneous, different nodes may have different conflicting objectives. Therefore, if each wireless node in a sub-tree tries to optimize its performance according to (3), the problem will become a Multi-Agent (MA) stochastic game with competing players [17]. It results in a non-stationary environment leading to divergence in finding an optimal solution for the network (sub-tree). To avoid this problem and achieve convergence, we first develop team objectives instead of individual optimization goals. (4) gives the overall team objectives as a linear weighted sum of all objectives.

$$\overline{QoS}(c) = \sum_{QoS \in Q_{obj}} \omega_{QoS} \times \widehat{QoS}(c), \quad (4)$$

where $\widehat{QoS}(c)$ is the normalized value of the QoS metric in Q_{obj} and ω_{QoS} is its weight. Consequently, the optimization problem will be as follows.

$$\begin{array}{lll} optimise & \overline{QoS}(c) & \\ c \in C & & \\ s.t. & QoS_j(c) \succeq QoS_j^{req} & \forall\ QoS_j \in Q_{cnt} \end{array}, \quad (5)$$

where $\overline{QoS}(c)$ is the overall performance to be optimized. To solve this optimization problem, the wireless nodes within a sub-tree need to collaborate to optimize the developed team objectives. For this, the nodes need to constantly exchange control packets to inform one another about their actions and the outcomes. To avoid this overhead, we convert the MA problem to a single-agent by making the parent node in a sub-tree a single-agent decision maker, solving the optimization scenario in (5), and informing the nodes about the action they must take.

In the rest of this paper, as an example and without loss of generality, we assume three common performance metrics as reliability, latency, and energy consumption for each node. From the application's point of view, it is the end-to-end performance metrics that should be considered. However, the end-to-end performance metrics can be derived by combining the single hop level performance metrics [2]. Thus, we consider three QoS metrics in the single hop level as Packet Loss Ratio (PLR), Latency (L), and Energy consumption (E) which are defined for a typical node n_i as follows. These metrics are calculated in a predefined time episode.

$\boldsymbol{PLR_i}$ is the number of arrived packets at node n_i that are failed to be delivered to the parent node over the total number of n_i's arrived packets during the time episode. Note that n_i's arrived packets can be the generated packets by n_i or the packets it receives from its children to forward to the parent node. The failed packets are counted regardless of whether the packets are transmitted (they may be transmitted but failed to be delivered to the receiver or may be discarded due to the arrival of newer packets).

$\boldsymbol{L_i}$ is defined as the time between the packet arrival at n_i and its delivery to the parent node. It is averaged over all delivered packets from n_i during the time episode. Note that latency is calculated only for data packets that are successfully delivered to the parent node.

$\boldsymbol{E_i}$ is the average energy consumed by n_i to transmit a packet to the parent node and receive its ACK during the time episode. Generally, the energy consumption (E) of a node during a time episode is a function of the node's activities such as transmitting, receiving, switching between radio modes, and being in sleep mode in all timeslots in that time episode. In this work, we use the energy model described by (6), w.l.o.g ignoring the details such as energy consumed during mode switches or the sleep periods.

$$E = E_s \times TxN_s + E_f \times TxN_f \quad (6)$$

In (6), E_s is the energy consumption of a node for transmitting a packet successfully (sending the packet and receiving its ACK), while E_f is the energy consumption of the node when it transmits a packet but fails to receive the ACK packet. TxN_s and TxN_f are the number of successful and failed transmissions, respectively. Clearly, $TxN_s = 1$, while TxN_f depends on the *maxR* value. E_s and E_f are computed by

$$E_s = P_{TX} \times T_{TX} + P_{RX} \times T_{ACK} \quad , \tag{7}$$

$$E_f = P_{TX} \times T_{TX} + P_{RX} \times T_{to} \quad , \tag{8}$$

where P_{TX} and P_{RX} are the radio power consumption of the node in transmitting and receiving states, respectively. T_{TX}, and T_{ACK} are the time durations of data packet transmission and receiving the ACK. In case the ACK packet is not sent by the receiver or it is lost, the standard specifies a time-out (T_{to}) until which the transmitter node continues listening to the channel for the ACK. Afterward, the data packet transmission is considered a failure by the transmitter, and the retransmission process is initiated. According to the standard, $T_{to} \simeq T_{ACK}$. Thus, $E_s \simeq E_f$, and $E = E_s \times TxN$, where $TxN = TxN_s + TxN_f$.

Accordingly, E_i depends on the number of transmissions denoted by TxN_i and the power consumption profile of the node. In the rest of this paper, we use TxN_i as a representation of E_i in the MAC layer.

As stated in (4), the normalized values of these metrics are used.

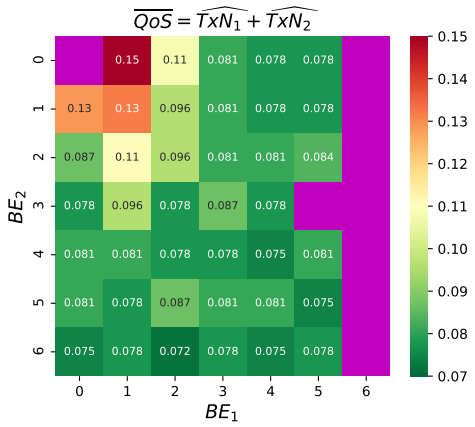
B. Need for dynamic and non-equal settings

In many CSMA/CA configuration schemes for TSCH networks, static and equal configurations are set for all nodes as BE^{min}, BE^{max}, and *maxR*. In the following, we show that this approach results in CSMA/CA settings that are far from optimal for heterogeneous nodes with diverse QoS requirements in dynamic environments. Here, we use a simple subtree including only two nodes (n_1 and n_2) running TSCH CSMA/CA. The packet arrival periods are 200ms and 250ms for n_1 and n_2, respectively. By incident, their first transmissions start at the same timeslot so they experience collisions. Let energy consumption of both nodes have to be optimized subject to constraints on *PLR* of n_1 (PLR_1), and L of n_2 (L_2) as $PLR_1 \leq 0.1$, and $L_2 \leq 100ms$.

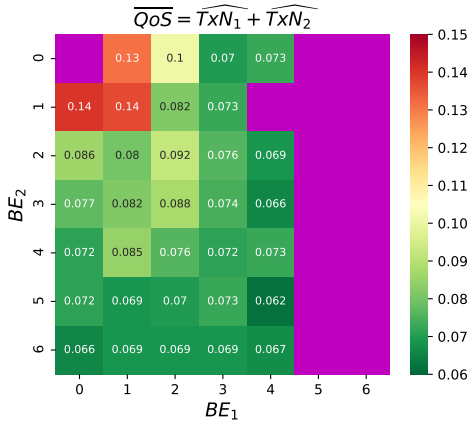
From (4) and (5), we have

$$\begin{aligned} \underset{c}{minimise} \quad & \overline{QoS}(c) = \widehat{TxN_1} + \widehat{TxN_2} \\ s.t. \quad & PLR_1(c) \leq 0.1 \\ s.t. \quad & L_2(c) \leq 100ms \end{aligned} \tag{9}$$

For simplicity of showing results, it is assumed that $BE_i^{min} = BE_i^{max} = BE_i$ for node n_i. Through Monte-Carlo simulations, the performance of the nodes for different values of BE_1 and BE_2 is derived. Fig. 3(a) shows $\overline{QoS}$ defined in (9) versus BE_1 and BE_2, wherein $\overline{QoS}$ values are reported per $[BE_1, BE_2]$. For values of BE_1 and BE_2 that the constraints are not satisfied, the box is in purple color, and $\overline{QoS}$ values are not reported.



(a) $\overline{QoS}$ vs BE_1 and BE_2 when packet arrival periods are 200ms and 250ms for n_1 and n_2, respectively



(b) $\overline{QoS}$ vs BE_1 and BE_2 when packet arrival periods are 100ms and 120ms for n_1 and n_2, respectively

Fig. 3. Overall performance ($\overline{QoS}$) versus configuration settings showing unsuitability of equal and static settings for dynamic heterogeneous networks

Equal configuration settings ($BE_1 = BE_2$) for heterogeneous nodes are not the optimal solution. The optimum configuration settings are $BE_1 = 2$ and $BE_2 = 6$. As another case, the traffic load in the network is increased by setting the packet arrival period of n_1 and n_2 to 100ms and 120ms, respectively. Observing the results in Fig. 3(b), it is shown that the network gets far from the optimal behavior, and optimal configuration settings change as $BE_1 = 4$ and $BE_2 = 5$. Consequently, a self-optimization framework has to be designed to provide adaptive and per-node CSMA/CA configuration settings to guarantee distinctive QoS.

V. Learning-based TSCH CSMA/CA configuration

To approach the problem, we first choose QL as a representative model-free RL algorithm, since it conducts systematic trial and error in the unknown environments that are too complex to build a model. To implement QL, agent, state, action, and reward functions are defined for our problem. Then, the convergence time of QL algorithm as a limiting factor for the networks with large configuration space is discussed.

Neural networks are leveraged in QL algorithm to accelerate the convergence in order to overcome the limiting factor. To realize this, DQN framework is presented which keeps the same definition of state, action, and reward functions in QL, while employing neural networks to approximate Q-values.

A. QL-based TSCH CSMA/CA configuration

Here, we first define QL elements as an agent, states, actions, and a reward function to create a QL model for the configuration of TSCH CSMA/CA parameters according to QoS demands. The state s_t is an element of configuration space C including all nodes' configuration settings. The action a_t is the step size of adjusting each configuration parameter, which can be ± 1 or zero. A parent node as an agent implements QL algorithm to select an appropriate CSMA/CA configuration based on experience gained from agent-environment interactions. The proposed QL-based MAC protocol features a QL-based scheme that adaptively regulates the configuration settings to maximize the overall performance according to the defined constraints and objectives. Having Q_{obj} and Q_{cnt}, the reward is defined as,

$$r_{t+1} = \begin{cases} -3 \times N & \exists i \mid \ PLR_i > PLR_{dis} \\ -\sum |\widehat{QoS}_j(c) - \widehat{QoS}_j^{req}| & \forall j \mid \ QoS_j \in Q_{cnt} \ \ \& \\ & \quad QoS_j(c) \prec QoS_j^{req} \\ \frac{1}{\widehat{QoS}(c)} & otherwise \end{cases}, \tag{10}$$

where N is the number of nodes connected to the parent node. Our most important goal is to configure CSMA/CA parameters in a way so that all nodes have the opportunity to deliver their packets to the parent node, meaning that the scheme is expected to go for settings that avoid very high PLR values. If data packets of a node fail to reach the parent node, other QoS metrics such as latency cannot be truly measured. If any of the nodes gets stuck in a severe situation (disconnection) and cannot deliver its data packets to its parent node ($PLR \approx 1$), the agent will be harshly punished and gets the largest negative reward. Here, such a severe situation is translated as $PLR > PLR_{dis}$, where PLR_{dis} is the threshold of PLR after which the link is considered disconnected. If this is not the case, then the agent checks whether all constraints are satisfied. If some of the constraints are not met, the agent is punished by a negative reward which is the sum of the gap between the normalized QoS values and their corresponding requirements. As stated in Sec. IV-A, we assume three QoS metrics to evaluate the performance in a sub-tree. So, the largest possible negative reward is $-3 \times N$. If none of the first two cases holds, the inverse of the overall performance is considered as a positive reward.

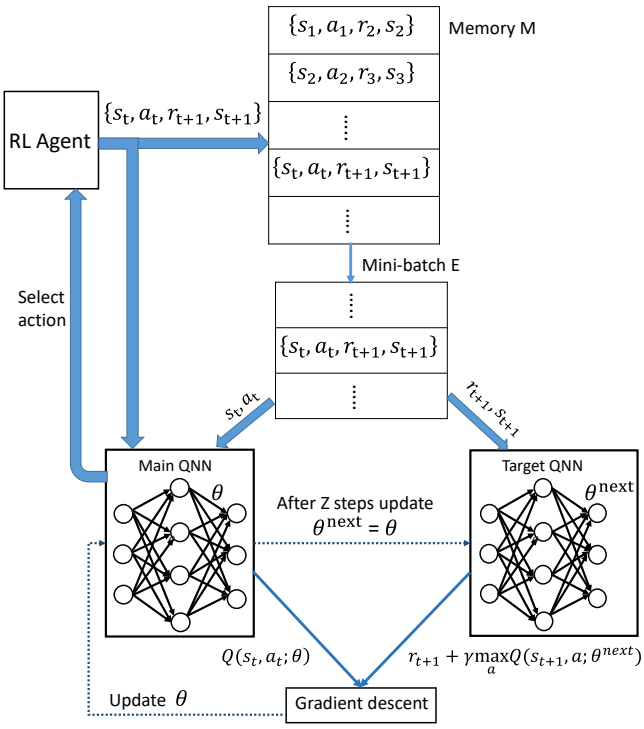


Fig. 4. The structure of the developed DQN framework

B. DQN Framework for TSCH CSMA/CA configuration

By increasing the number of nodes and/or configuration settings per node, the state-action space expands exponentially. To get a rough idea, suppose a sub-tree with only three nodes whose three CSMA/CA parameters per node have to be configured, resulting in 9 configuration parameters. There are 312 possible configurations per node [2] resulting in approximately $312^3 \times 3^9 \approx 6 \times 10^{11}$ state-action pairs. Each state-action value must be explored at least once for convergence to the optimal solution. If each time step duration is 100ms for exploration, the convergence of RL takes more than one year. Therefore, when a problem has a large state-action space we can no longer represent QL as an efficient solution. Instead of using a large table to represent Q, we can approximate the state-action value function using explored experience as a ground truth.

$$Loss(\theta, E) = \frac{1}{|E|} \times \sum_{e_t \in E} \big[Q(s_t, a_t; \theta) - (r_{t+1} + \gamma \max_a Q(s_{t+1}, a, \theta^{next}))\big]^2 \tag{11}$$

Therefore, we present DQN which adopts QNN as the approximation function for mapping states to actions in order to improve the convergence speed significantly. In QNN, when the parameters of the neural network are updated based on the explored experience in one state, the Q estimates for other similar states are changed too. Therefore, parameterization facilitates the generalization of the experience. We first started with the original DQN algorithm described in Section II-B2. However, it showed divergent behavior on several scenarios.

It is a well-known instability problem in DQN which has two causes [9]. First, a single experience is used in the algorithm to calculate loss function (2), leading to instability. Second, one QNN is utilized to approximate both predicted and the ground truth Q-values making high bias. To stabilize the DQL algorithm, we modify the original DQN by applying two key ideas [9]. First, instead of training QNN with a single experience at time t (e_t), multiple experiences are pooled together and stored in memory M for batch training. A mini-batch denoted by E including $|E|$ random experiences is chosen from M to compute the loss function for a round of training. Thus, Mean Square Error (MSE) is employed as a loss function instead of (2) as follows [18].

The second idea is the use of two different QNNs called "target QNN" and "main QNN". The structure of them are the same, but the parameters are different, in which the parameters of the main QNN (θ) are up-to-date, while those of the target QNN (θ^{next}) are older. After Z steps, the parameters of the target QNN will be replaced by that of the main QNN. In the computation of loss function in (11), target QNN approximates $Q(s_{t+1}, a, \theta^{next})$. Then, $r_{t+1} + \gamma \max_a Q(s_{t+1}, a, \theta^{next})$ is used as ground truth for Q-values, while main QNN approximates $Q(s_t, a_t; \theta)$ that is used as predicted Q-values.

Fig. 4 shows our DQN framework structure whose process is described in Algorithm 1. It is implemented in the parent node as an RL agent. The algorithm starts with the initialization of the environment, first-time step, initial state, and other learning parameters. The memory is empty at the beginning and is filled with explored experiences over time in a FIFO manner. Then the parent node gets the specifications of the nodes including their packet arrival period, QoS objectives, and their constraints (line 4). This is done by receiving control packets including this information from its children. In line 5, the parameters of the learning machine are set to their initial values. Afterward, there is a while loop that iterates over time steps while the node's specification and QoS requirements do not change.

We utilize $\epsilon-$greedy policy as the behavior strategy to explore and exploit in action selection with ϵ gradually decreasing by rate ϵ_{dec} from 1 to ϵ_{min}. The agent executes the selected action and enters the next state s_{t+1} in line 12 which determines new configuration settings. The agent broadcasts it to the nodes and waits till the end of the time step to evaluate the QoS of all nodes in lines 13-14. The reward function is calculated by the agent from (10) in line 15, and experience $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ obtained from the interaction between the agent and the environment at time step t is stored in M as stated in line 16. When at least $|E|$ experience is added to the experience memory M (line 17), training the QNNs starts by selecting the mini-batch E randomly from M and feeding it to the two QNNs (lines 18-20). Then the approximated values out of the main QNN and the target QNN are used to compute loss function in line 21. The loss function is minimized in gradient descent operation with a learning rate of β [18] to update θ values. The θ values of the target QNN are updated to the latest θ parameters of the main QNN after Z time steps in lines 26-27.

Algorithm 1: DQN algorithm

```
1  Initialise the environment, time step t ← 1, get initial state s_1
2  Initialise parameters θ randomly, and set θ_next ⟵ θ
3  Initialise M, E, and Z
4  Get nodes' Spec., Q_obj, and Q_cnt
5  Initialise β, ε_min, ε_dec, γ, and set ε ← 1
6  M ← ∅          /* Memory is empty at the beginning */
7  while (!Changed(Spec., Q_obj, Q_cnt )) do
8      if random < ε then
9          a_t ← random(A)                  /* exploration */
10     else
11         a_t ← arg max_a Q(s_t, a; θ)     /* exploitation */
12     s_{t+1} ←Execute(a_t) /* distribute new config. to
           nodes                                          */
13     Wait for one time step
14     EvaluateQoS()
15     r_{t+1} ← RewardFunction() /* using (10)          */
       /* store the experience e_t in M                  */
16     M.append(s_t, a_t, r_{t+1}, s_{t+1})
17     if |M| ≥ |E| then
           /* select a mini-batch of size |E| from M     */
18         E ← RandomSelect(M)
           /* approximate Q(s, a, θ)                     */
19         Q(s_t, a_t, θ) ← Main QNN.forward
           /* approximate Q(s, a, θ^next)                */
20         Q(s_{t+1}, a_t, θ^next) ← Target QNN.forward
21         Loss ← LossFunction() /* using (11)          */
22         θ ← GradientDescent(Loss, β)
           /* decrease epsilon                           */
23         ε ← max(ε − ε_dec, ε_min)
24     s_t ← s_{t+1}
25     t ← t + 1
26     if t % Z=0 then
27         θ^next ← θ
28 if MajorChange then
29     Go to line 1
30 else
31     ε, ε_dec, |E| ← UpdateExplorationScheme()
32     Go to line 6
```

The parent node always keeps track of new control information it receives from its nodes. Once it detects a change in the nodes' specification or QoS demands, it reacts by altering the usual procedure of the learning process. In Algorithm 1, the main loop breaks when such changes are detected. Note that it is very inefficient and loss of experience if the trained QNNs are reset after any change in the sub-tree. If the change is not major, the already trained networks can be used as a base and get new training by exploring the new situation. Our experiments show that it hugely speeds up the convergence of the algorithm upon changes leading to the higher reactivity of the mechanism. Thus, what the parent node does is evaluating the observed change and adapting the learning process accordingly. If the change is minor such as a change in the QoS constraints of a node, it is accommodated by resetting the exploration parameters namely ϵ, ϵ_{dec}, $|E|$, flushing the experience memory (M) and continuing with the

already trained QNNs. Going to line 6, the algorithm flushes M, and then starts to learn again using new explorations. If the change is detected as a major change such as a node joining or leaving, a different state-action space is resulted for which the input and output layers of QNN need to be changed. Thus, the agent needs to reset everything and executes from the scratch (line 1).

VI. Performance Evaluation

In this section, we discuss experiments performed to evaluate the performance of our proposed DQN framework as a run-time self-configuration platform in a TSCH network. The convergence time of the configuration mechanism to derive QoS-satisfying configurations is an important factor. Also, we aim at investigating the performance of the mechanisms upon state changes at run-time, and checking whether the network can reconfigure itself quickly to satisfy diverse and dynamic QoS consistently. QL and pure CSMA/CA with equal CSMA/CA settings are implemented and tested; their QoS results are compared with the DQN framework.

We adopt a neural network with two hidden layers, wherein the number of neurons for the first and the second layers is 30 and 10, respectively. The activation function is ReLU [18], and MSE in (11) is used as the loss function in the gradient descent method with $\beta = 0.05$. The value of ϵ in the ϵ-greedy algorithm is initially set to 1 and is decreased at a rate of 0.005 ($\epsilon_{dec} = 0.005$) every time step until its value reaches 0.01 ($\epsilon_{min} = 0.01$) . The discount factor, γ is set to 0.99. The size of the experience memory, M is set to 800, the mini-batch size is set to $|E| = 80$, and the target QNN is updated every 50 training steps ($Z = 50$). Each training step should be long enough to observe the performance of packet delivery of all nodes. We consider one second as the length of a training step including 100 shared timeslots (timeslot length is 10 ms). After each training step, the parent node calculates the reward and broadcasts new configuration settings to all children nodes. The disconnection threshold for PLR in the reward function is set to $PLR_{dis} = 0.97$ in all the simulations. In all setups, it is assumed that the wireless links in the PHY layer are ideal with a packet reception ratio of 100%. It means that if there is no collision in accessing a shared timeslot, the packet is assumed to be received by the parent node successfully. This assumption is made in the performance evaluations to isolate the results from the impact of channel deficiencies.

We first consider a network including three nodes transmitting their data to the parent node. It is assumed that n_1, n_2, and n_3 have a packet to send to the parent node every 50, 70, and 130ms. n_1, n_2, and n_3 have to minimize their *PLR*, latency (L), and energy consumption (*TxN*), respectively, subject to $PLR_i \leq 0.3$ for $1 \leq i \leq 3$. In other words, $Q_{obj} = \{PLR_1, L_2, TxN_3\}$ and $Q_{cnt} = \{PLR_1, PLR_2, PLR_3\}$. In this setup, we set $BE^{min} = BE^{max} = BE$, in which *BE* can vary in range $[0, 7]$, and $maxR = 7$ for all nodes. Accordingly, the parent node runs DQN to tune BE_1, BE_2, and BE_3 after any change to optimise the overall performance as $\widehat{PLR_1} + \widehat{L_2} + \widehat{TxN_3}$.

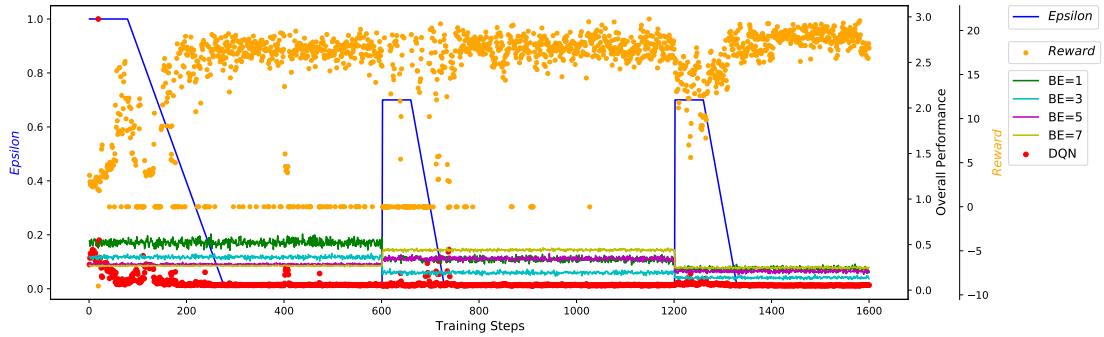
To investigate the adaptability of our DQN framework to the changes in the network, we change nodes' packet arrival periods after 10 minutes and 20 minutes. After the first 10 minutes, the packet arrival period of the nodes n_1 and n_3 changes to 150 and 90ms, respectively. Then packet arrival period of n_3 changes after the second 10 minutes to 270ms. Since these changes are not considered as major changes, we update the exploration scheme by setting of $\epsilon = 0.7$, $\epsilon_{dec} = 0.01$, and $|E| = 60$ after the nodes' packet arrival period changes. Thus, we tested the cases in which we reset ϵ to its initial value of one. However, the convergence speed is higher when we use a lower value (0.7 in our implementation with a higher decrease rate and smaller mini-batch).

Fig. 5(a) compares the overall performance of our proposed DQN framework during the described 30 minutes of network conditions with the overall performance of pure CSMA/CA by which all nodes get equal configuration settings ($BE = 1, 3, 5$, and 7). The main observation is that the overall performance achieved by the DQN framework converges to the best-achieved results in several minutes. In dynamic IoT systems, this time is reasonably accepted. Fig. 5(b) shows selected values of BE_1, BE_2, and BE_3 versus the training step. It is observed that the selected configuration settings when DQN framework converges are different for the nodes, and it changes when the network state changes over time. The network can react to the changes in the traffic load generated by the nodes quickly (around 3 minutes), and reconfigure the nodes to the new configuration settings.

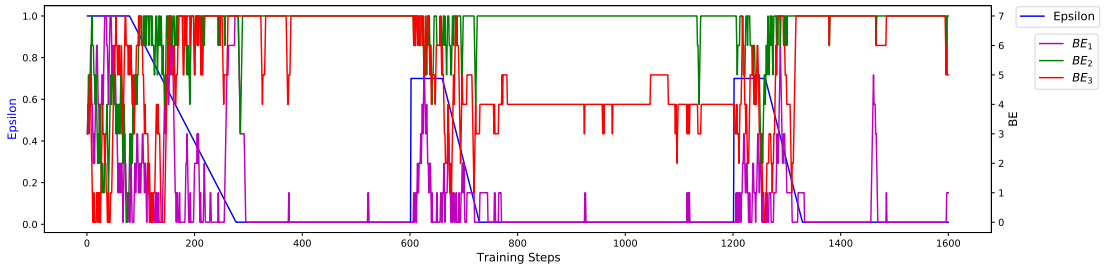
Constraint performance metrics are shown in Fig. 5(c). It is observed that they satisfy their corresponding requirements ($PLR_i \leq 0.3$) in the convergence zone. Note that the probability of exploration is 0.01 in the convergence zone, consequently, the agent may still select a random action with a probability of 0.01. It makes the network a little far off near-optimal values which are visible in Fig. 5.

To investigate the strength of DQN compared to QL in terms of converging, we implemented QL for the same setup and runs the QL for the first 10 minutes when the packet arrival period are 50, 70, and 130ms. Fig. 6 shows the overall performance as $\widehat{PLR_1} + \widehat{L_2} + \widehat{TxN_3}$ when the agent runs QL with the same learning parameters used in DQN. As shown, QL cannot converge to the results achieved by DQN at the same exploration time. It is intuitively expected since QL has to explore all state-action pairs or it cannot converge to an optimal policy. Instead, DQN does not need to explore all state-action pairs since it approximates state-action values using the explored state-action pairs.

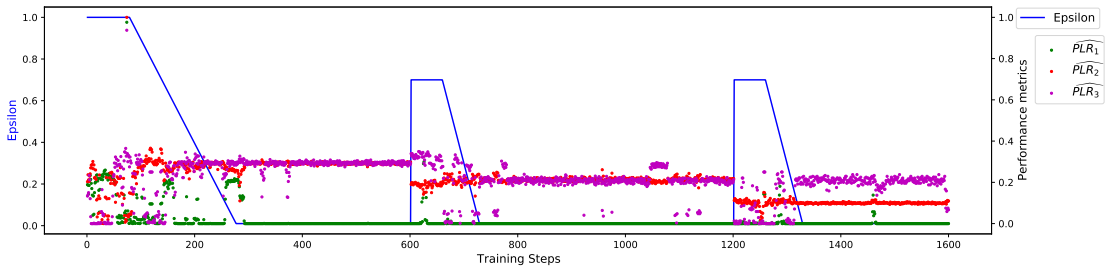
In the second setup, a new node with a packet arrival period of 170ms (n_4) joins the network in the previous setup when the packet arrival periods of the existing nodes are 150, 70, and 270ms. It is assumed that latency of n_4 is also to be minimized. Thus, L_4 is added to the objectives set, Q_{obj}. Accordingly, the parent node detects a major change and resets the Algorithm to start learning from scratch. Fig. 7 gives the results for this setup showing that the network can recover itself in a few minutes,



(a) Comparison between the overall performance when using DQN with pure CSMA/CA



(b) BE values of the nodes when using DQN



(c) Constraint performance metrics when using DQN

Fig. 5. The results of the network including three nodes with packet arrival periods as 50, 70, 130ms, which changes to 150, 70, 90ms after 10 minutes, and then changes to 150, 70, and 270ms after 20 minutes, when $Q_{obj} = \{PLR_1, L_2, TxN_3\}$ and $Q_{cnt} = \{PLR_1, PLR_2, PLR_3\}$ using DQN and pure CSMA/CA

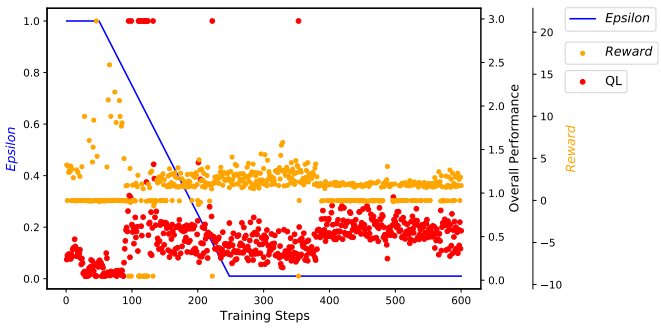
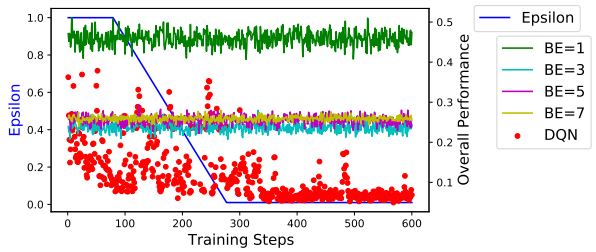


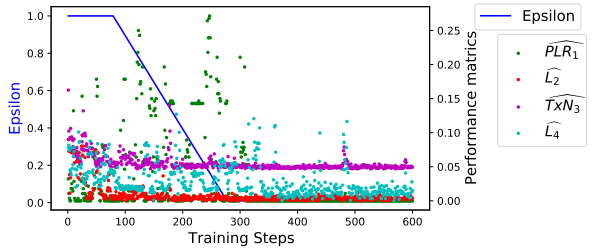
Fig. 6. Overall performance of the network including three nodes with packet arrival periods as 50, 70, 130ms when $Q_{obj} = \{PLR_1, L_2, TxN_3\}$ and $Q_{cnt} = \{PLR_1, PLR_2, PLR_3\}$ using QL

and it converges to the QoS-satisfying configurations. Fig. 7(a) presents the overall performance, $\widehat{PLR_1} + \widehat{L_2} + \widehat{TxN_3} + \widehat{L_4}$. It is observed that the network has converged to its final configuration after around 350 learning steps (seconds). Also, the observed overshoots around learning step 500 are random explorations that RL performs. Fig. 7(b) shows the individual performance objectives behavior. Fig. 7(c) confirms that all constraints are met meaning that $PLR_i \leq 0.3$ for $i = 1, 2, 3$. Moreover, Fig. 7(d) shows how a suitable value is assigned to BE_4 for the new joined node in a way that the overall performance is minimized and the constraints are met.

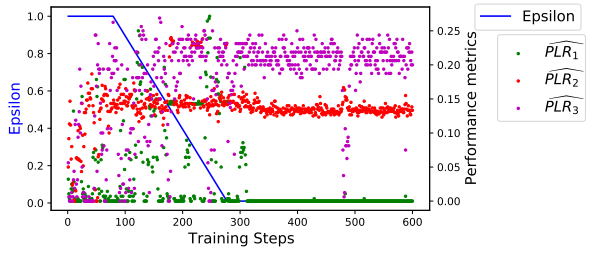
In the last setup, we use the first setup including three nodes when the packet arrival periods are 50, 70, and 130ms. The same objective performance metrics are considered with no constraints. Thus, $Q_{obj} = \{PLR_1, L_2, TxN_3\}$ and $Q_{cnt} = \varnothing$. The assumptions in the first setup as $BE^{min} = BE^{max}$ and $maxR = 7$ are released. Configuration parameters, BE_i^{min}, BE_i^{max}, and $maxR_i$ can get any value in the range [1,7]. Increasing the number of configuration settings makes the state-action space larger resulting in complex QNN. Fig. 8 shows the overall performance for this setup. Here, we implement pure CSMA/CA with four typical value sets for [BE^{min}, BE^{max},



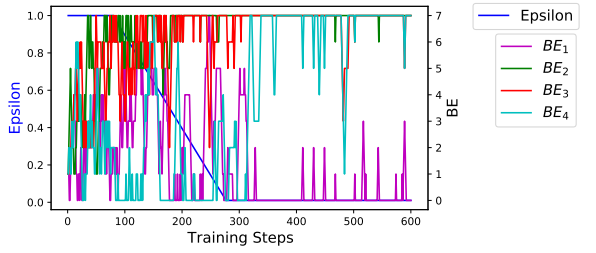
(a) Comparison between the overall performance when using DQN with pure CSMA/CA



(b) Objective performance metrics when using DQN



(c) Constraint performance metrics when using DQN



(d) BE values of the nodes when using DQN

Fig. 7. The results of the network including four nodes with packet arrival periods as 150, 70, 270, 170ms when $Q_{obj} = \{PLR_1, L_2, TxN_3, L_4\}$ and $Q_{cnt} = \{PLR_1, PLR_2, PLR_3\}$ using DQN and pure CSMA/CA

maxR] which are stated in Fig. 8. As shown, our DQN can converge to the best setting in several minutes compared to pure CSMA/CA. This is while, the configuration parameters are increased to 9 parameters resulting in a huge configuration space, and making complex neural networks with many inputs and outputs.

Comparing the results for all setups reveals the effectiveness of our DQN framework in selecting the right TSCH CSMA/CA settings from a very large configuration space to meet diverse and dynamic QoS requirements at run-time. Moreover, it can recover itself and reconfigure the parameters of the nodes in a reasonable time when the network status changes.

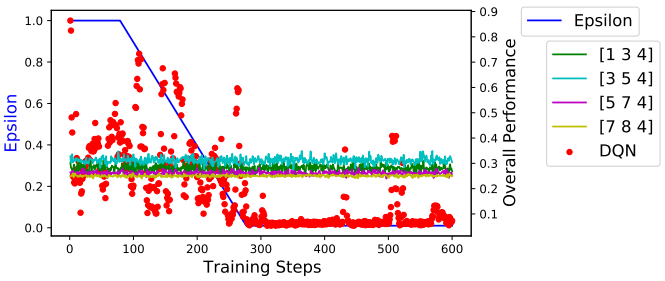


Fig. 8. Overall performance of the network including three nodes with packet arrival periods as 50, 70, 130ms and 9 configuration parameters when $Q_{obj} = \{PLR_1, L_2, TxN_3\}$ and no constraint using DQN and pure CSMA/CA

VII. Running DRL on IoT Embedded Devices

Running DRL algorithms on IoT devices requires storing some parameters and performing arithmetic operations to train the Neural Networks (NNs) and process the data generated by the IoT devices. For a NN with n hidden layers, and specifications of i, $\{h_k\}$, and o, indicating the number of neurons in the input layer, hidden layer k-th $(k = 1, .., n)$, and output layer, respectively, the total number of trainable parameters ($|\theta|_{NN}$) is given by (12).

$$|\theta|_{NN} = i \times h_1 + \sum_{k=1}^{n-1} h_k \times h_{k+1} + h_n \times o + \sum_{k=1}^{n} h_k + o \quad (12)$$

Moreover, the number of floating point multiplications (MLP_{NN}) in its operation is calculated as follows.

$$MLP_{NN} = i \times h_1 + \sum_{k=1}^{n-1} h_k \times h_{k+1} + h_n \times o \quad (13)$$

To have a rough estimation of the computation and memory load of the algorithms, consider a DRL setup running on a TSCH network in Section VI, including two NNs (the main and target). The configuration settings are the inputs, and there are three actions per configuration setting. Thus, $i = 3 \times 3$ (three nodes and three configuration parameters per node), $o = 3 \times i$, $h_1 = 30$, and $h_2 = 10$. Then, the number of NN parameters to store for both NNs is 1814. To store only trainable parameters as single precision floating points, $8KB$ of memory capacity is needed. Also, it performs 1680 floating points multiplications, ignoring any multiplications that are associated with the activation function, reward function, etc. This leads to a rather high computation time ($>$ 10ms) and a borderline memory using the general purpose embedded processors on typical IoT devices. Thus, running DRL algorithms on resource-limited IoT embedded devices is still a challenge for the research and industrial communities.

Recently, new research is being conducted to develop new approaches to tackle this issue and implement DRL algorithms on resource-constrained IoT devices [19]. On the one hand, efforts are put forth in developing next-generation embedded

processors capable of running efficient machine learning algorithms to be used for IoT edge devices. On the other hand, NN model optimizations are being explored (e.g., structure compression and quantization) in order to to reduce the load of deep learning-based techniques for embedded devices. An interesting example is "Tiny ML / Tensorflow Lite" [20], which runs machine learning at the core of IoT devices operating at low and ultra-low power.

Although running DRL algorithms presented in this work may be challenging for currently commercially available IoT nodes due to their limitation mainly in terms of floating-point computation power, it is highly anticipated that machine learning-capable embedded processors soon become available. Then the already developed learning-based optimization techniques are ready to play their significant role.

VIII. Conclusion

In this paper, we investigated the essence of dynamic and non-equal CSMA/CA configuration settings in a multi-hop TSCH network consisting of heterogeneous nodes with different packet arrival rates and diverse QoS demands. To address this problem, we developed a model-free RL algorithm called QL to set configuration parameters and adapt them at run-time according to the network changes. Due to the large state-action space, neural networks are adopted in our QL algorithm to approximate state-action values and create a self-optimization framework called the DQN framework. It is implemented in the parent node of a tree-based multi-hop TSCH network to configure CSMA/CA parameters of its children. Through simulations, it is verified that our DQN framework can provide a better self-optimization and adaptation platform for a TSCH network rather than pure CSMA/CA which uses equal configuration settings for heterogeneous nodes regardless of divers QoS. It is shown that our proposed DQN framework can react quickly to the changes in the network and tune the network configuration parameters to consistently meet the diverse and dynamic QoS in several minutes even for a large configuration space.

References

[1] "IEEE Standard for Low-Rate Wireless Networks," *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pp. 1–709, April 2016.

[2] H. Hajizadeh, R. Tavakoli, M. Nabi, and K. Goossens, "Approximated Pareto Analysis for Fast Optimization of Large IEEE 802.15.4 TSCH Networks," in *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*, 2020, pp. 1–7.

[3] M. Zlmmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "pTUNES: Runtime Parameter Adaptation for Low-power MAC protocols," in *2012 ACM/IEEE 11th International Conference on Information Processing in Sensor Networks (IPSN)*, 2012, pp. 173–184.

[4] J. Shi and M. Sha, "Parameter Self-Configuration and Self-Adaptation in Industrial Wireless Sensor-Actuator Networks," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 658–666.

[5] T. Winter and P. Thubert and A. Brandt and J. Hui and R. Kelsey and P. Levis and Kristofer S. J. Pister and R. Struik and JP. Vasseur and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," *RFC*, vol. 6550, pp. 1–157, 2012.

[6] Sutton, Richard S. and Barto, Andrew G., *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998. [Online]. Available: http://www.cs.ualberta.ca/%7Esutton/book/ebook/the-book.html

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[8] C. J. Watkins and P. Dayan, "Q-Learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, " Human-level Control Through Deep Reinforcement Learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] Y. Chen, Y. Liu, M. Zeng, U. Saleem, Z. Lu, X. Wen, D. Jin, Z. Han, T. Jiang, and Y. Li, "Reinforcement Learning Meets Wireless Networks: A Layering Perspective," *IEEE Internet of Things Journal*, vol. 8, no. 1, pp. 85–111, 2020.

[11] H. Xu, X. Liu, W. G. Hatcher, G. Xu, W. Liao, and W. Yu, "Priority-aware Reinforcement-learning-based Integrated Design of Networking and Control for Industrial Internet of Things," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4668–4680, 2020.

[12] M. Abbasi, A. Shahraki, M. J. Piran, and A. Taherkordi, "Deep Reinforcement Learning for QoS provisioning at the MAC layer: A Survey," *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104234, 2021.

[13] X. Zhang, M. Lei, C. Wang, and M. Zhao, "A Transmission and Backoff Method Based on Deep Reinforcement Learning for Statistical Priority-based Multiple Access Network," in *2021 IEEE 94th Vehicular Technology Conference (VTC2021-Fall)*, 2021, pp. 1–5.

[14] Y. Zhao, J. Hu, K. Yang, and S. Cui, "Deep Reinforcement Learning Aided Intelligent Access Control in Energy Harvesting Based WLAN," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 11, pp. 14 078–14 082, 2020.

[15] Z. Jiang, S. Zhou, and Z. Niu, "Distributed Policy Learning Based Random Access for Diversified QoS Requirements," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[16] S. Duquennoy, B. Al Nahas, O. Landsiedel, and T. Watteyne, "Orchestra: Robust Mesh Networks Through Autonomously Scheduled TSCH."

[17] Y. Yang and J. Wang, "An overview of multi-agent reinforcement learning from game theoretical perspective," *arXiv preprint arXiv:2011.00583*, 2020.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[19] Z. Zhang and A. Z. Kouzani, "Implementation of DNNs on IoT devices," *Neural Computing and Applications*, vol. 32, pp. 1327–1356, 2020.

[20] [Online]. Available: https://www.tinyml.org/

Hamideh Hajizadeh received the B.Sc. degree in electronics engineering and the M.Sc. degree in communication systems engineering both from Tehran University. She is currently working towards her Ph.D. in electrical and computer engineering at the Eindhoven University of Technology (TU/e), Eindhoven, the Netherlands.

Majid Nabi (Member, IEEE) received the B.Sc. degree in computer engineering from Isfahan University of Technology, Isfahan, Iran, in 2001, the M.Sc. degree in computer engineering from Tehran University, Tehran, Iran, in 2007, and the Ph.D. degree in electrical and computer engineering from Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands, in 2013.

He is currently an Assistant Professor at the Department of Electrical Engineering, TU/e, and Isfahan University of Technology. His research interests include efficient and reliable networked embedded systems, low-power wireless sensor networks, and Internet-of-Things.

Kees Goossens (Member, IEEE) has a BSc in computer science from the University of Wales (1988), and a PhD from the University of Edinburgh (1993). In his thesis he investigated the formal verification of hardware, in particular by using semi-automated proof systems in conjunction with formal semantics of hardware description languages such as ELLA and VHDL. He continued this work at several other universities before joining Philips Research in the Netherlands in 1995. At Philips he worked on behavioural synthesis for high-throughput video processing, then on on-chip communication protocols and memory management. Until 2010, at Philips/NXP Semiconductors Research he led the team that defined the Aethereal network on chip for consumer electronics, where real-time performance and low cost are major constraints. He was also part-time full professor at the Delft university of technology from 2007 to 2010, and is currently full professor at the Eindhoven university of technology, where his research focusses on composable (virtualised), predictable (real-time), low-power embedded systems, supporting multiple models of computation. He is also system architect at Topic Embedded Products, working on real-time dependable dynamic partial reconfiguration in FPGAs. He is editorial board member for the ACM Transactions on Design Automation of Electronic Systems (TODAES), associate editor for the Springer Journal of Design Automation of Embedded Systems (DAEM), and was guest editor for several special issues on networks on chip. He is author on 25 patents, and published four books, 100+ articles, with four paper awards. His 2003 paper was selected as one of the 30 most influential papers of 10 years of the DATE conference. He is or was steering committee member of ACSD, NOCS, MPSOC, and TPC member of CODES+ISSS, CRTS, DATE, DSD, ECRTS, FPL, ICPP, INA-OCMC, OMHI, NoCArc, PARMA, QVVP, ReConFig, RTAS, SAMOS, SDR, SOC, and VLSI-SOC, etc.