CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	RECCE: Deep reinforcement learning for joint routing and scheduling in time-constrained wireless networks
Authors	Chilukuri, Shanti;Pesch, Dirk
Publication date	2021-09-22
Original Citation	Chilukuri, S. and Pesch, D. (2021) 'RECCE: Deep reinforcement learning for joint routing and scheduling in time-constrained wireless networks', IEEE Access, 9, pp. 132053-132063. doi: 10.1109/ACCESS.2021.3114967.
Type of publication	Article (peer-reviewed)
Link to publisher's version	10.1109/access.2021.3114967
Rights	© 2021, the Authors. This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0 - https://creativecommons.org/licenses/by/4.0/
Download date	2026-11-04 10:26:33
Item downloaded from	https://hdl.handle.net/10468/14349



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 29, 2021, accepted September 13, 2021, date of publication September 22, 2021, date of current version October 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3114967

RECCE: Deep Reinforcement Learning for Joint Routing and Scheduling in Time-Constrained Wireless Networks

SHANTI CHILUKURI[1], (Member, IEEE), AND DIRK PESCH[2], (Senior Member, IEEE)
[1]Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (deemed to be University), Visakhapatnam 530045, India
[2]School of Computer Science and IT, University College Cork, Cork, T12 K8AF Ireland

Corresponding author: Shanti Chilukuri (chilukurishanti@gmail.com)

This work was supported in part by the European Union's Horizon 2020 Research and Innovation Programme through the Marie Sklodowska Curie EDGE COFUND Grant under Agreement 713567, and in part by the Science Foundation Ireland through the CONNECT Centre under Grant 13/RC/2077 and through the ENABLE Spoke under Grant 16/SP/3804. This work was done when Shanti Chilukuri was at University College Cork, Ireland.

ABSTRACT Time Division Multiple Access-based Medium Access Control protocols tend to be the choice for wireless networks that require deterministic delay guarantees, as is the case in many Industrial Internet of Things (IIoT) applications. As the optimal joint scheduling and routing problem for multi-hop wireless networks is NP-hard, heuristics are generally used for building schedules. However, heuristics normally result in sub-optimal schedules, which may result in packets missing their deadlines. In this paper, we present RECCE, a deep REinforcement learning method for joint routing and sCheduling in time-ConstrainEd networks with centralised control. During training, RECCE considers multiple routes and criteria for scheduling in any given time slot and channel in a multi-channel, multi-hop wireless network. This allows RECCE to explore and learn routes and schedules to deliver more packets within the deadline. Simulation results show that RECCE can reduce the number of packets missing the deadline by as much as 55% and increase schedulability by up to 30%, both relative to the best baseline heuristic. RECCE can deal well with dynamic network conditions, performing better than the best baseline heuristic in up to 74% of the scenarios in the training set and in up to 64% of scenarios *not* in the training set.

INDEX TERMS Routing and scheduling, multihop networks, deep reinforcement learning, time constraints, IIoT.

I. INTRODUCTION

There is an ever-increasing deployment of wireless networks for Industrial Internet of Things (IIoT) applications such as electrical power management, pollution and gas monitoring and factory automation. Some of these applications (gas or fire detection, control systems in factories etc.) have strict packet delivery time-constraints. The ramifications of packets missing their deadlines (mandated by the application QoS requirements) can be quite serious for such applications (e.g., machine failure in manufacturing), and may even lead to injury to humans. While several factors contribute to packet delay, the Medium Access Control (MAC) delay is a major factor. This delay can be made more deterministic by using a Time Division Multiple Access (TDMA) approach at the MAC layer as in many IoT protocols such as WirelessHART [1] and IEEE 802.15.4-TSCH [2].

The time sensitive operation of TDMA hinges on *schedules* which assign time slots and channels to nodes that transmit or receive data. The end-to-end packet delay in a multi hop TDMA wireless network depends on such schedules. A *feasible schedule* is one in which all packets are delivered within their deadlines. Since optimal slot allocation in TDMA networks is NP-hard [3], heuristics such as those proposed in [4]–[6] are generally employed to find the schedules. However, heuristics generally result in sub-optimal schedules that may cause packets missing their deadlines.

Generally, the goal of scheduling in time-constrained networks is to find a feasible schedule for a given network scenario, but it may not always be possible to find a feasible schedule for some network scenarios. In [7], the authors propose a scheduling goal to minimize the number of packets

The associate editor coordinating the review of this manuscript and approving it for publication was Aasia Khanum.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

missing the deadline and the time by which they miss it. This goal does not just find a feasible schedule when possible or minimize the average packet delay, but also *tries to minimize the extent by which a schedule falls short of being feasible when it is not possible to find a feasible schedule*. As this is a finer goal when finding schedules for time-constrained networks, we focus on this goal in this paper.

Reinforcement Learning (RL) and more recently, Deep Reinforcement Learning (DRL), are attractive approaches to solve problems that can be modelled as Markov Decision Processes (MDP). RL is a learning method where the RL agent learns a policy to maximize the expected cumulative reward [8]. During training, the RL agent performs random actions based on the system state and observes the reward. After training for sufficient time, the agent can arrive at the ''optimal'' policy π^*, which gives the best action to take in any given state, such that the expected cumulative reward is maximized. When the state or action space is large, simple RL may require extensive training and memory. Deep RL is an approach to RL where the state-value function or the policy is found by function approximation using deep neural networks. RL and DRL have been shown to yield good performance for decision-making problems, especially scheduling [7], [9] and routing [10], [11].

In this paper, we present a Deep **RE**inforcement Learning-based scheme for Joint Routing and S**C**heduling in Time-**C**onstrained N**E**tworks (called *RECCE*). Unlike other heuristics which consider the same criterion (e.g., laxity, relative deadline) for scheduling in each TDMA time slot, RECCE leverages DRL to apply varying criteria for forwarding packets in each time slot. While this approach is similar to RLSchedule proposed in [7], a major difference is the choice of routes followed by packets. While RLSchedule always forwards packets along the best (shortest) route, RECCE explores alternate routes while forwarding packets. This helps RECCE find schedules and routes that reduce delay due to queuing and node conflicts. Simulation results show that in spite of not sticking to the optimal route, RECCE performs much better than baseline heuristics in terms of the percentage of packets missing the deadline and their delay budget overshoot.

II. RELATED WORK

Scheduling for real-time data transmission has been studied extensively (e.g. [4]–[6] and papers therein). The proposed solutions generally have the goal of minimizing the average packet delay or maximizing the schedulability, which is the percentage of feasible schedules found. In this paper, our main focus is on packets missing their deadlines and we attempt to reduce their number and delay budget overshoot, in case a feasible schedule, where all packets meet their deadline, is not possible for a given network scenario. In case feasible schedules are possible, we aim to identify the one that minimizes the average packet delay. This goal is similar to the one proposed in [7], but our approach is different from that in [7], as discussed below.

Scheduling can be modelled as an MDP and RL (or DRL for large and/or continuous action or state spaces) has been used with success for job scheduling in data processing clusters [9] or link-scheduling in cellular networks [12], [13], for example. The DRL agent design depends on the network type (e.g. cellular networks, mesh networks, vehicular ad hoc networks) and the goal in mind (e.g. minimize end-to-end delay, maximize link utilization). As we consider time-constrained applications in wireless networks (e.g. control applications in the IIoT) with the goal to minimize packets missing the deadline, our research is different from the above-cited studies.

The authors of [14] explore a deep deterministic policy gradient (DDPG)-based approach for centralized scheduling of data sources connected to a software defined network with heterogeneous link access rates. The centralized controller determines the pacing rates of the sources based on their deadline-driven data transfer requests. We consider centralized scheduling with a much finer time granularity (TDMA slots). In [7], the authors propose RLSchedule, a deadline-aware centralized TDMA scheduling mechanism based on Deep Reinforcement Learning. Our goal and scheduling options are similar to that addressed in [7] but we consider exploration of alternate routes, as discussed in Section I. Simulation results show that this approach meets the scheduling goal better than RLSchedule.

There have been several solutions for optimal routing based on reinforcement learning. An excellent survey of these techniques is given in [15]. Of the work discussed in this paper, [10] and [11] use RL to decide which pre-computed path to follow to meet their scheduling goal (reducing the average call holding time and reducing the energy consumed). While these papers consider the choice of a route just once per flow, [16] makes this choice at every node along the route, using Dijkstra's algorithm to determine the maximum delay bound for any link. During the run-time phase, safe state space exploration is done by eliminating links that exceed the required delay bounds. RECCE also uses RL to choose between pre-computed routes at each node, potentially letting it switch between paths at any time slot. However, RECCE considers TDMA with constant link delay (one time slot) and combines scheduling with routing to deliver data within the deadline. To the best of our knowledge, the use of RL for joint TDMA scheduling and routing in time-constrained wireless networks has not been explored till now.

III. REINFORCEMENT LEARNING - BASICS

In this section, we present a very brief overview of RL and then the DRL algorithm used by us – Proximal Policy Optimization [17].

In reinforcement learning, a policy π is essentially a mapping from a system state s to an action a (if π is deterministic) or the probability of an action a (if π is stochastic). The goal of an RL agent is to learn the optimal policy π^* where the expected cumulative reward is maximized for every state. This is done by performing random actions in each state

during training and observing the reward. There are two main approaches to RL —

- Value-based methods, where the Q-value (the state-value function) of each state and action is calculated during training. The optimal policy is determined by the action that gives the maximum Q-value in a given state.
- Policy gradient methods, where the policy is directly parameterized and the agent learns the optimal policy using deep neural networks and gradient descent (or ascent).

Value-based methods (e.g., Q-learning or deep Q-learning) are simple but sometimes, the Q-function can be too complex to model. In such cases, policy gradient methods are preferred as they converge to a good policy for a variety of environments. Generally, policy gradient methods also converge faster than deep Q-learning. For policy gradient methods, the loss function is given by the following expectation [8]:

$$L^{PG}(\theta) = \hat{\mathbb{E}}[log\pi_\theta(a_t|s_t)\hat{A}_t] \tag{1}$$

Here, $\hat{A}_t$ is the advantage at time step t for a policy π with a vector of parameters θ. A positive value of $\hat{A}_t$ means that action a_t is better than others in state s_t. Using gradient ascent on the above loss function pushes the policy more towards actions that give more reward. However, if the step-size for the loss function is too small, training can be very slow and if it is too large, training can be unstable. PPO [17] is an actor-critic method that alleviates this issue.

Actor-critic methods are a third category of RL algorithms that use both the above techniques to find the optimal policy. There are several variants of actor-critic methods but in a simple actor-critic method, the actor is the policy and the critic is the state-value function. After the actor selects an action a_t for a state s_t, the critic evaluates the policy. This evaluation is in the form of an error and is given by [8]:

$$\delta_t = rew_{t+1} + dV(s_{t+1}) - V(s_t) \tag{2}$$

Here, rew_{t+1} is the immediate reward, d is the discount factor for the reward and V is the current value function implemented by the critic. If the error is positive, then a_t is considered a good action in state s_t, otherwise not. PPO avoids large policy updates (on $L^{PG}(\theta)$ in Equation 1) at any step by using a clipped surrogate objective loss function ([18]). Limiting the policy update stabilizes the training and avoids convergence to local optima.

PPO has been proven to give good results while taking less time for training as the environment becomes more complex, compared to other policy gradient or actor-critic methods. For more details on PPO, please refer to [17].

IV. NETWORK MODEL

We consider a network model similar to the one in [5] and [7]. The network has a set $\mathcal{N}$ of nodes, each with a half-duplex radio that can transmit or receive in any of the M channels. The notation used is summarized in Table 1. Links between nodes may have weights to denote energy, link quality etc. We consider that link weights denote the inverse log of the link packet delivery ratio (PDR), which is the ratio of packets correctly delivered to the destination to those generated at the source. This makes the total length of the route proportional to the end-to-end packet delivery probability. Note that while TDMA eliminates any packet loss due to collisions, the delivery probability on any link can still be less than 1.0 due to channel impairments such as fading.

TABLE 1. Notation.

Symbol	Meaning
$\mathcal{N}$	set of nodes
M	number of channels
$\mathcal{F}$	set of flows
σ_i	start slot of the i^{th} flow
$deadline_i^{rel}$	relative deadline of the i^{th} flow
p_i	priority of the i^{th} flow
δ_i	period (in slots) of the i^{th} flow
γ	number of routes considered for routing
$\mathcal{R}_i$	ordered set of the top γ shortest routes of i^{th} flow
$\mathcal{D}_k^t$	the set of transmissions in the transmission queue of node k at time slot t
$deadline_i^{abs}$	absolute deadline of the i^{th} flow
h_i^j	number of hops in route $\overrightarrow{R_i^j}$
u_i	node at which transmission τ_i is queued
v_i^j	next-hop node of transmission τ_i along the j^{th} route
hr_i^j	remaining hops of transmission τ_i along the j^{th} route
tr_i	time remaining (in slots) until deadline for transmission τ_i
α_{min} and α_{min}	lower and upper limits of the harmonic period
β	ratio of deadline to period
T_ψ	hyper-period of scenario ψ
$\overrightarrow{S^t}$	system state vector at slot t
$\mathcal{A}$	set of possible actions
π	policy of the DRL agent
π^*	optimal policy for the DRL agent
ω_{missed}	reward for each missed packet
ω_{lost}	reward for each packet not delivered in a hyper-frame
Ψ	set of scenarios

In our network model, randomly selected nodes act as sources and generate packets of data either periodically or upon some event. The generated packets may take multiple hops to be delivered to other network nodes (destinations). The network has $|\mathcal{F}|$ *flows* (periodic or event-based) at any given time, where each flow f_i in the set $\mathcal{F}$ consists of a packet that traverses from its source to destination. A flow f_i is characterised by:

- its source and destination nodes,
- the release time σ_i, which is the TDMA slot at which a packet is generated at its source,
- the relative deadline $deadline_i^{rel}$ which is the number of slots after σ_i by which a packet has to reach its destination,
- a priority $p_i \in W$ based on the application to which the flow belongs. Here, W is the set of whole numbers. As the value of p_i increases, the priority decreases. Hence, a flow with p_i=0 has maximum priority.
- a period δ_i (harmonic, in slots), if the flow is periodic and
- an ordered set of routes $\mathcal{R}_i$, which consists of the top γ shortest routes. Each member of the set $\mathcal{R}_i$ is

the route $\overrightarrow{R_i^j}$, where $j \in \{1, .., \gamma\}$. The routes in $\mathcal{R}_i$ are found by using a shortest path algorithm such as Yen's algorithm [19].

The slot number by which a flow has to reach its destination is its *absolute deadline*. This is denoted by $deadline_i^{abs}$ and is given as:

$$deadline_i^{abs} = deadline_i^{rel} + \sigma_i \tag{3}$$

Route $\overrightarrow{R_i^j}$ has h_i^j hops from source to destination. At any time slot t, the network has a set of *transmissions* that can be scheduled. Transmission τ_i represents a packet at a particular hop and —

- consists of a packet belonging to the i^{th} flow at a node u_i,
- depending on the route j chosen, has the next hop v_i^j and the remaining number of hops to the destination hr_i^j and
- at time slot t, has time remaining until deadline tr_i = $deadline_i^{rel}$ - (t-σ_i).

We assume that the flows have harmonic periods ([20]) randomly chosen between $2^{\alpha_{min}}$ and $2^{\alpha_{max}}$. The relative deadline $deadline_i^{rel}$ of each flow is assumed to be some fraction (β) of its period. Hence, $deadline_i^{rel} = \beta * \delta_i$, where $\beta \in (0, 1]$. We also assume that the controller and the nodes are synchronized using some standard mechanism (for example, using messages in the control plane). This is a necessary condition for all TDMA-based medium access control.

When scheduling is performed centrally, the scheduler (e.g., network controller) has the *scenario* (the network topology and flow) information. Based on this, it creates a route for each flow and a schedule for the network using these routes. For such a network scenario (denoted by ψ), the *hyper-period* T_ψ is the least common multiple of the individual periodic flow periods. It is enough if the scheduler finds a schedule for the hyper-period, as this schedule can just be repeated for each hyper-frame until the network scenario changes. Without RECCE, the route chosen is normally the shortest route and the schedule is found using some heuristic (e.g., Earliest Deadline First (EDF), Deadline Monotonic (DM)).

In most real-time TDMA protocols such as WirelessHART, the time slot is of sufficient duration to allow the transmission of a packet and its acknowledgement between adjacent node pairs. When the links are lossy, reliable data delivery is normally performed through retransmissions following negative acknowledgements or timeouts. However, retransmissions until the packet is successfully delivered may result in indeterminate delay. To increase reliability and still have bounded delay, some kind of over-provisioning is generally considered in TDMA systems by either —

- allotting more than one – but a small fixed number – of time slots for each transmission or
- making each time slot large enough to accommodate a limited number of retransmissions between adjacent node pairs.

This over-provisioning is a trade-off potentially resulting in larger end-to-end delay for increased reliability. Hence, we do not consider retransmissions in this version of RECCE. Nevertheless, the second design choice above (longer time slots) can be seamlessly accommodated by RECCE.

V. BASIC PRINCIPLES OF RECCE

RECCE uses Deep Reinforcement Learning to explore alternate routes and scheduling strategies and learn the best scheduling and routing policy. To do this, during training, the DRL agent observes the system (network) state at time slot t, denoted by the vector $\overrightarrow{S^t}$, takes actions from a set of possible actions $\mathcal{A}$, and observes the resulting reward. The action taken, together with any packets generated or delivered, result in a new state of the network for the next time slot and this process is then repeated.

RECCE uses an actor-critic algorithm based on Proximal Policy Optimization described in Section III, and learns in episodes. Each episode in RECCE starts with slot 0 of a hyper-frame and ends with slot $T_\psi - 1$. The DRL agent moves to the end of the episode in steps, with each step representing a time slot. As shown in Algorithm 1, the length of the episode depends on the periods of the flows in the chosen scenario ψ, as T_ψ is the least common multiple of all the flow periods. As it explores the state and action spaces, the DRL agent refines its policy π, which is a map of the action to be taken at a given state for maximum expected cumulative reward. After running enough episodes, the agent arrives at an *optimal* policy π^*, which provides the best action in any state so as to maximize the cumulative reward.

A. STATE SPACE

In RECCE, each node $n_k \in \mathcal{N}$ in the network has a tuple of features (denoted by $(x1_k^t, x2_k^t, x3_k^t, x4_k^t)$), that represent the state of transmissions queued at a given time slot t. Assuming that $\mathcal{D}_k^t$ is the set of transmissions in the transmission queue of n_k at time slot t, these features are:

1) $x1_k^t = \frac{1}{|\mathcal{D}_k^t|}$, the inverse of the transmission queue length, if $|\mathcal{D}_k^t| > 0$, -1 otherwise.
2) $x2_k^t = \min_{\forall \tau_i \in \mathcal{D}_k^t}(tr_i)$, the minimum time remaining.
3) $x3_k^t = \min_{\forall \tau_i \in \mathcal{D}_k^t, \forall j \in \{1,..,\gamma\}}(hr_i^j)$, the minimum hops remaining, and
4) $x4_k^t = \min_{\forall \tau_i \in \mathcal{D}_k^t}(p_i)$, the maximum priority, since we consider $p_i = 0$ to be greatest priority.

The system (network) state at time slot t, $\overrightarrow{S^t}$, is the concatenation of the node features of all the $|\mathcal{N}|$ nodes. Hence, the length of $\overrightarrow{S^t}$ is $4|\mathcal{N}|$.

B. ACTION SPACE

During training, the DRL agent tries different criteria at each time slot in the hyper-period T to choose up to M transmissions. The criteria applied for choosing these transmissions are based on popular heuristics as in [7]. This can be based on one of the following six factors:

1) the relative deadline (Deadline Monotonic(DM))
2) the absolute deadline (Earliest Deadline First (EDF))

3) the ratio of relative deadline to the total number of hops (Proportional Deadline (PD))
4) the ratio of remaining time to the remaining number of hops (Earliest Proportional Deadline (EPD))
5) the value of remaining time minus the remaining number of hops (Least Laxity First (LLF))
6) the node features $(x1_k^t, x2_k^t, x3_k^t, x4_k^t)$

In each of the first five cases, transmissions that yield the minimum from one of these factors are chosen to be transmitted. When the last criterion (node features) is considered, the node with minimum features is chosen based on the criterion. A node feature tuple $(x1_k^t, x2_k^t, x3_k^t, x4_k^t)$ is less than another tuple $(x1_l^t, x2_l^t, x3_l^t, x4_l^t)$ if $x1_k^t \leq x1_l^t$ & $x2_k^t \leq x2_l^t$ & $x3_k^t \leq x3_l^t$ & $x4_k^t \leq x4_l^t$. Then, the transmission with minimum time remaining at this node is chosen. In case more than one transmissions have the same value for any chosen criterion, the transmission with highest priority among them is chosen to be scheduled.

At each time slot, in addition to different scheduling criteria, RECCE considers a next hop node v_i^j which can be the next hop of u_i on any of the top γ routes. Thus, RECCE explores at most γ next hops at each hop of a packet. The set of possible actions at each time slot is denoted by $\mathcal{A}$ and combining the number of scheduling and routing options, $|\mathcal{A}| = 6\gamma$.

C. REWARD

Since RL agents are trained to maximize the expected cumulative reward, the reward design plays a major role in the design of an RL agent. With our scheduling goal in mind, the agent gathers rewards at each time step as follows:

- For every packet delivered to the destination, a reward inversely proportional to the end-to-end delay is collected.
- If a packet is delivered within the hyper-frame in which it is generated but not within its deadline, a large negative reward ω_{missed} is collected.
- For every packet generated but not delivered within the same hyper-frame, a larger negative reward ω_{lost} is collected, where $\omega_{lost} \ll \omega_{missed}$.

This reward design helps the agent to *find schedules and routes such that the number of packets delivered within the deadline is maximized and the end-to-end delay is minimized.*

VI. RECCE IN PRACTICE

As DRL requires considerable time for training the agent, it may not be practically feasible to train the agent on actual running network deployments. We propose training the agent offline as in [7] and [21]. To do this, the network controller collects information about the frequently seen scenarios that are gathered in the set Ψ. Then, as proposed in [7], the agent may be trained in one of the following ways:

- The same scenario $\psi_i \in \Psi$ is considered for each training episode to arrive at a *customized policy* π_i^* for this scenario.
- Alternately, if a random scenario from Ψ is considered for each training episode, a *generalized policy* π^* can be arrived at. This can then be applied to any scenario in the set Ψ.

Algorithm 1 RECCE(Ψ)

Input: Number of nodes $|\mathcal{N}|$, Number of channels M, Network adjacency graph E and Flow information $\mathcal{F}$ for each scenario in Ψ

Output: Generalized Policy model π^* for the set Ψ

$j = 0, \quad t = 0$
{/*j is a loop variable to count the number of updates and t is the timeslot in the hyperframe*/}
Choose a random scenario ψ from Ψ
while $j <$ *max_updates* **do**
  $k = 0$
  {/*k is a loop variable for batch updates*/}
  while $k <$ *batch_size* **do**
    Calculate the node features to get the network state $\overrightarrow{S}^t$.
    Perform action a_t from $\mathcal{A}$ and calculate reward as per PPO
    Increment k, t
    if $t = T_\psi$-1 **then**
      $t = 0$
      Choose a random scenario ψ from Ψ
    end if
  end while
  Update policy π as per PPO
  Increment j
end while

Since RECCE's actions are based on popular scheduling heuristics and the top γ routes, we expect it should perform better than or at least as well as the best performing heuristic for all scenarios considered in training. However, building a general policy for a set of scenarios using DRL may result in some loss of scheduling performance. To check how much this is, we compared the performance (average over 100 non-contiguous TDMA cycles) of the two above approaches (titled RECCE-custom and RECCE-general) with that of the best baseline heuristic. Figure 1 shows the performance of the above two approaches in meeting the scheduling goal for a small sample of ten scenarios. It can be seen that a customized policy per scenario performs better than or as well as the best baseline heuristic for all network scenarios, in terms of the percent of packets delivered within the deadline. A generalized policy, on the other hand, can perform better than or as well as the best baseline heuristic in 9 out of the 10 cases in terms of percent of packets delivered within the deadline.

However, a building customized policy has some disadvantages. As discussed in [7], building a customized policy for each scenario requires more memory at the controller to store all the policies. Also, a customized policy may not work well

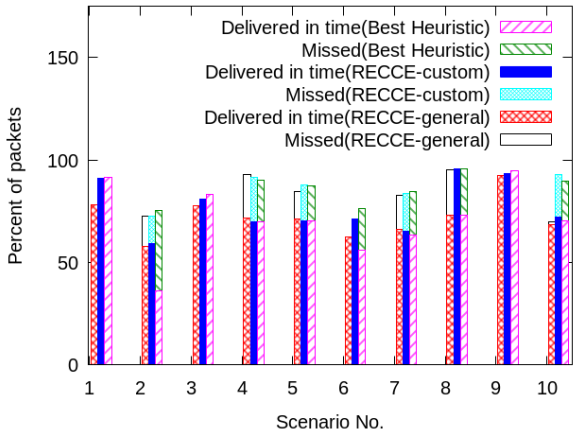


FIGURE 1. Performance of a customized policy vs a generalized policy.

for scenarios *not* in set Ψ. A generalized policy, on the other hand, occupies much less space at the controller as only a single policy needs to be stored and can also be applied to scenarios not in Ψ with some success.

A network controller may detect a changed scenario because of:

- a change in the network topology, due to the dynamic nature of a wireless network caused by changes in wireless connectivity or some limited node mobility or
- due to the occurrence of event-based flows.

Hence, we focus on the generalized policy approach for the rest of this paper. In case the network controller encounters new scenarios which are very different from the one in Ψ and the existing policy cannot give satisfactory performance, the agent can be retrained using the new set of scenarios to revise the policy.

VII. SIMULATION RESULTS

We evaluated RECCE for different network scenarios. Since we focus on a generalized policy for a set of scenarios Ψ, we created seven different sets of scenarios, each with 100 different network scenarios (with different topologies and traffic flows) as shown in Table 2.

TABLE 2. Scenario sets considered for simulation.

Scenario set (Ψ)	$\lvert\mathcal{N}\rvert$	M	$\lvert\mathcal{F}\rvert$	α_{min}	α_{max}	β	PDR	Path length (hops)		
								Mean	Median	Variance
1	10	1	4	4	4	0.5	0.5 to 1.0	3.26	3	1.04
2	10	1	4	4	4	0.5	0.7 to 1.0	3.20	3	1.12
3	10	2	4	4	4	0.5	0.5 to 1.0	3.27	3	1.1
4	10	2	4	4	4	0.5	0.7 to 1.0	3.16	3	1.11
5	20	2	8	5	5	0.5	0.5 to 1.0	4.21	4	3.09
6	20	2	8	5	5	0.5	0.7 to 1.0	4.2	4	3.87
7	50	4	15	4	5	0.75	0.7 to 1.0	5.37	5	6.45

Each scenario consisted of $|\mathcal{N}|$ nodes randomly placed in an area of 100m x 100m. Flows started and ended at randomly chosen nodes in the network and the network links were attached with weights equal to the log of the inverse of the PDR, which is generated randomly. The PDR ranged from 0.7 to 1.0 in four scenario sets and from 0.5 to 1.0 in the other three scenario sets, representing networks which have more lossy channels. For each flow f_i, an ordered set of routes $\mathcal{R}_i$ (the top γ routes in terms of the maximum end-to-end packet delivery ratio) were found using Yen's algorithm [19].

In reality, the period δ and the ratio of deadline to period β depend on the application's QoS requirements. Since scheduling is most difficult for frequent data generation (small periods) and tight deadlines, we considered periods from 2^4 to 2^5 and a deadline to period ratio of 0.5 and 0.75 to prove the efficacy of RECCE.

Where applicable, we compared RECCE to five popular baseline heuristics — deadline monotonic, earliest deadline first, proportional deadline, earliest proportional deadline and least laxity first [5]. In addition, we also compared it to the conflict-free least laxity first (CLLF) scheme proposed in [5] and a random policy, where a random action is performed at each time slot. For all these scheduling schemes, the most optimal route in terms of end-to-end PDR was considered.

We used a custom simulator written in C++ to simulate the baseline heuristics. For training the DRL agent, we used OpenAI Gym [22]. Particularly, we modified and added to the PPO implementation in [23]. The same packet-level abstraction was used for the baseline and DRL implementations. All simulations were repeated for 100 simulation runs each and the average values are reported. Unless otherwise mentioned, the results presented are the overall results for the entire set of scenarios.

A. EFFECT OF THE NUMBER OF ROUTES (γ)

RECCE considers the γ best routes (in terms of PDR) for packet transmission. The work presented in [7] considers only the best route, while using DRL for scheduling. Exploring sub-optimal routes may result in more packet loss (due to channel conditions), but may also result in better delay by avoiding nodes with conflicts or long packet queues. To study how the number of routes effects the number of missed packets, delay and also the total number of packets delivered, we trained DRL agents varying γ from 1 to 3 for the same set of network scenarios.

Figure 2 shows the cumulative distribution function (CDF) of the average packet delay for different number of routes γ in the case of two sample scenario sets (1 and 2, from Table 2). These sets have a PDR ranging from 0.5 to 1.0 and 0.7 to 1.0 respectively. For $\gamma = 2$ and 3, it can be seen that in spite of possibly choosing sub-optimal routes (i.e., links with more packet loss), the percent of packets delivered in a hyper-frame is comparable to that using just the optimal route ($\gamma = 1$, as in RLSchedule [7]).

Figure 3 shows the CDF of the average packet delay for packets that miss their deadline with the number of routes γ for these scenario sets. It can be seen that exploring more routes improves the performance of the scheduler in terms of the average delay of the missed packets and the number of missed packets.

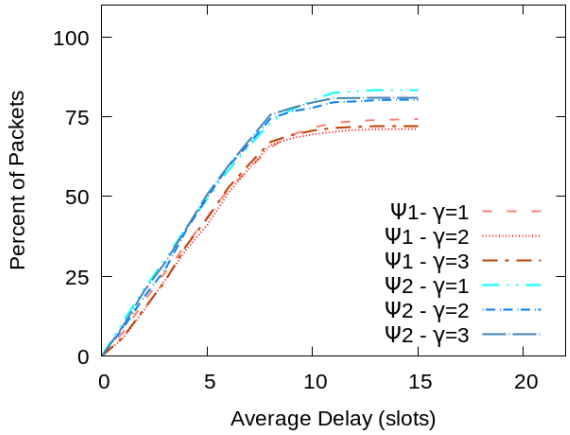


FIGURE 2. CDF of avg. Packet delay with the number of routes (N = 10, scenario set nos. 1 and 2).

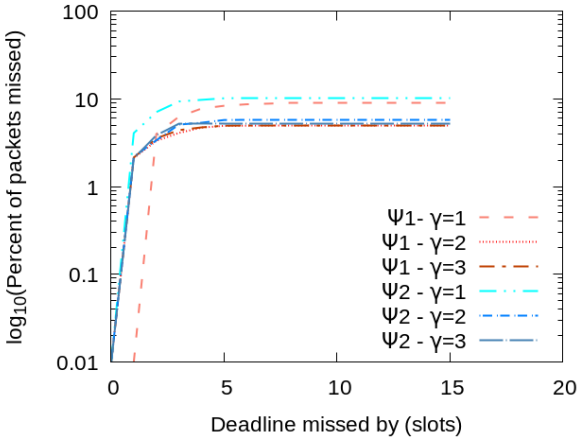


FIGURE 3. CDF of packet delay for missed packets with the number of routes (N = 10, scenario set nos. 1 and 2).

Normally, schedulability is taken to be the percentage of scenarios for which a feasible schedule can be found. However, since we considered links with probabilistic packet loss, different number of packets may miss the deadline for each hyper-period, though the same schedule and route are followed for a given scenario. To take this into account, we ran the simulation for multiple (100 in our simulations), non-contiguous hyper-periods per scenario. We define *schedulability for the entire scenario set as the number of hyper-periods in which there are no packets missing the deadline as a percent of the total number of scenarios in* Ψ *multiplied by the number of hyper-periods considered.*

Figure 4 gives this measure of schedulability for $\gamma \in \{1..3\}$. It can be seen from that exploring more routes is better in terms of schedulability too. For the rest of the paper we set γ to be 3, as this gives better performance in terms of packets missing the deadline and schedulability.

B. EFFECT OF THE NUMBER OF CHANNELS (M)

To study the effect of the number of channels on the performance of RECCE, we considered a small 10-node network

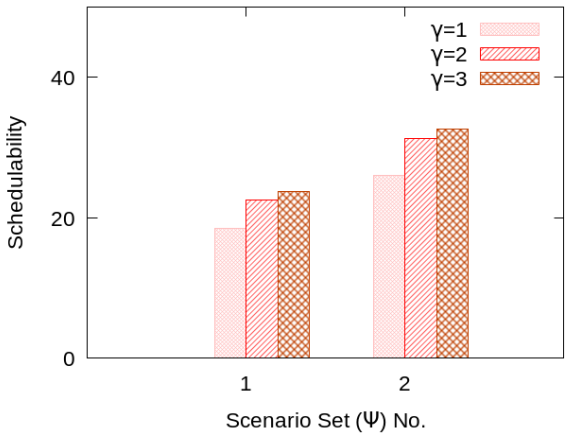


FIGURE 4. Schedulability with the number of routes, (N = 10, scenario set nos. 1 and 2).

with 4 flows. As the DRL agent has a state space proportional to the number of nodes $|\mathcal{N}|$, the time taken for training increases with the network size. In spite of this, RECCE is applicable to larger networks as well, as we propose offline training. Here, we consider small networks for proof of concept.

Figures 5 and 7 show the CDF of the average packet delay for Scenario set 2 with one channel and Scenario set 4 with 2 channels. The PDR ranges from 0.7 to 1.0 in both cases. Since the schedules obtained by two or more baseline heuristics are the same for some scenarios, their plots overlap.

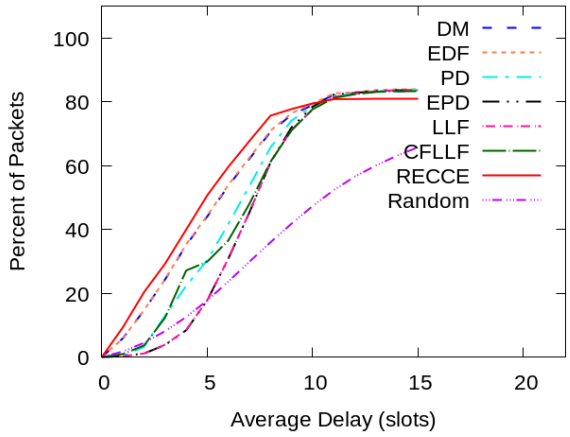


FIGURE 5. CDF of avg. Packet delay (N = 10 (Scenario set no.2), M = 1).

Figures 6 and 8 show the CDF of packet delay for packets missing the deadline for these scenarios. From these figures, it is evident that the percentage of packets missing the deadline is more for fewer number of channels for all scheduling policies. RECCE yields less packet delay with fewer number of packets missing their deadline compared to the best baseline heuristic (5.76%, compared to 12.78% given by the best heuristic for $M = 1$). The random policy performs the worst for both $M = 1$ and 2.

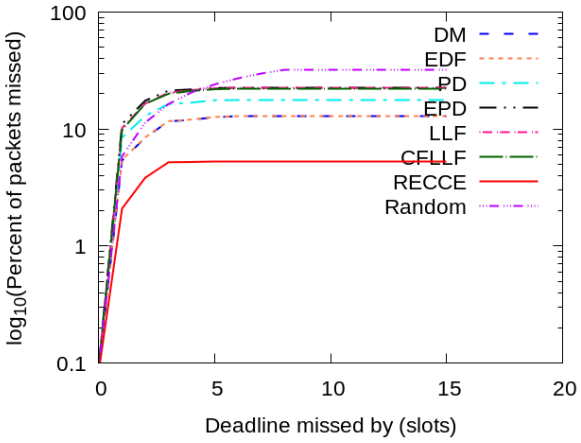


FIGURE 6. CDF of packet delay for missed packets (N = 10 (Scenario set no. 2), M = 1).

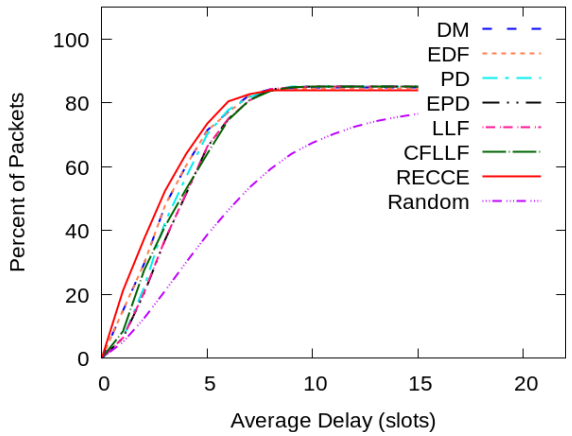


FIGURE 7. CDF of avg. Packet delay (N = 10 (Scenario set no.4), M = 2).

It can be seen in Figure 5 that the total percentage of packets delivered by RECCE within a hyper-frame is slightly lower (by 2.5%) than the heuristic that delivers the maximum percentage of packets. This is because RECCE may choose routes with more lossy links in some cases. For any set of scenarios, the percent of packets delivered within the deadline = (total percent of packets delivered − percent of packets missing the deadline). As the number of packets missing the deadline is improved by nearly 7% from Figure 6, RECCE is still better in terms of the total percent of packets delivered within the deadline.

C. EFFECT OF THE PACKET DELIVERY RATIO

RECCE explores routes other than optimal (in terms of packet delivery ratio) for packet delivery. While the goal of RECCE is to reduce the percent of packets missing the deadlines, following routes with more packet loss may raise a concern about the percentage of packets being delivered within a hyper-frame.

To study the effect of poor channel conditions and possible choice of lossy routes on the performance of RECCE, we considered two ranges of packet delivery ratios - one from 0.5 to 1.0 and the other from 0.7 to 1.0. For two network

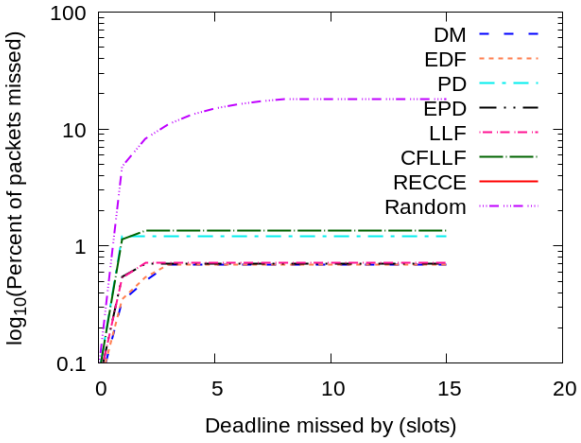


FIGURE 8. CDF of packet delay for missed packets (N = 10 (Scenario set no. 4), M = 2).

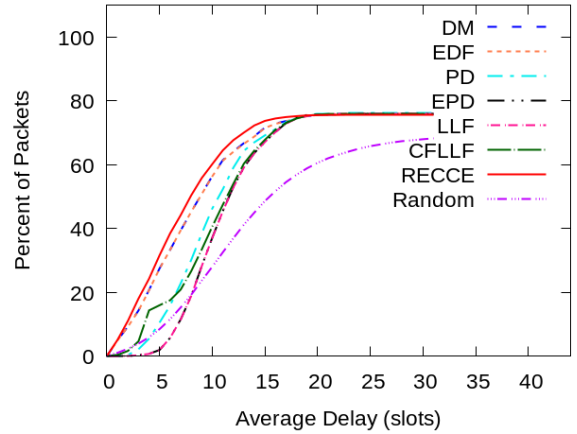


FIGURE 9. CDF of avg. Packet delay (N = 20 (Scenario set no.5), PDR from 0.5 to 1.0).

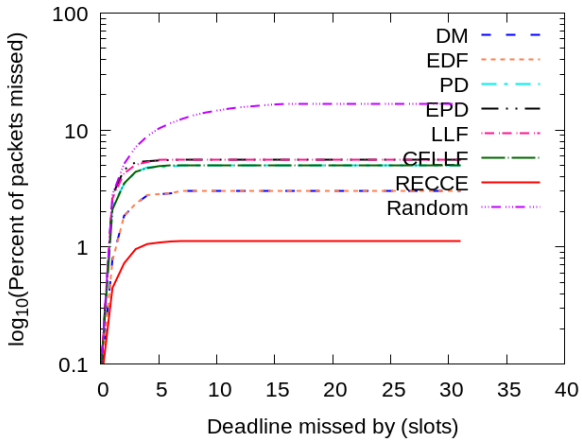


FIGURE 10. CDF of packet delay for missed packets (N = 20 (Scenario set no. 5), PDR from 0.5 to 1.0).

scenario sets (numbers 5 and 6 from Table 2) with 20 nodes, the CDF of packet delay for all the scenarios in Ψ is shown in Figures 9 and 11. It can be seen that while poor channel conditions reduce the number of packets delivered for all policies, RECCE performs better in terms of the average packet delay, while delivering almost equal percent of packets compared to other policies (which all use only the optimal

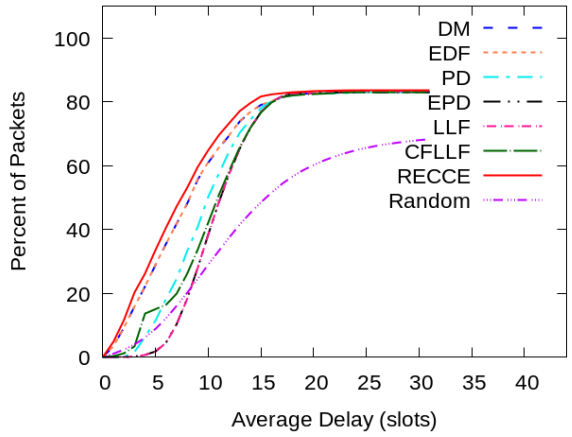


FIGURE 11. CDF of avg. Packet delay (N = 20 (Scenario set No.6), PDR from 0.7 to 1.0).

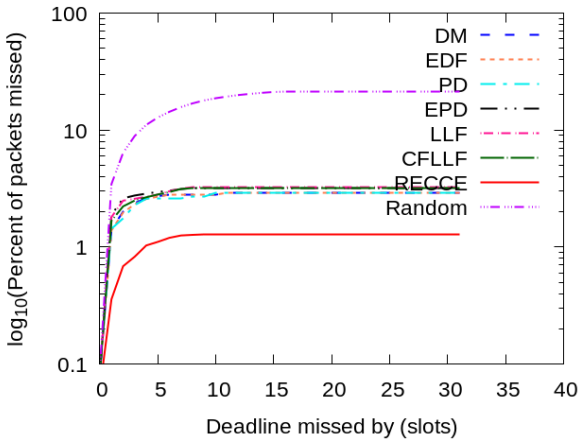


FIGURE 12. CDF of packet delay for missed packets (N = 20 (Scenario set no. 6), PDR from 0.7 to 1.0).

route). From Figures 10 and 12, it can be seen that the percent of packets missing their deadlines is much better with RECCE.

D. PERFORMANCE WITH UNSEEN NETWORK SCENARIOS

As discussed in Section VI, RECCE creates a generalized policy using a set of (most frequently seen) network scenarios Ψ for training. When the entire set of scenarios is considered, RECCE performs better than other baseline heuristics when the scheduler applies the policy to scenarios in Ψ. However, due to node mobility and dynamic channel conditions, the network topology may change or the set of flows may change due to event-based flows and the scheduler may encounter scenarios that are not in Ψ.

To evaluate the performance of RECCE for such scenarios unseen during training, we apply the policy created by the DRL agent to a different set of scenarios Ψ'. Scenarios in Ψ' have the same parameters (N, M, α_{min}, $alpha_{max}$, β and PDR range) as those in Ψ, but have different network graphs and flow end-points. Figure 13 depicts the relative performance

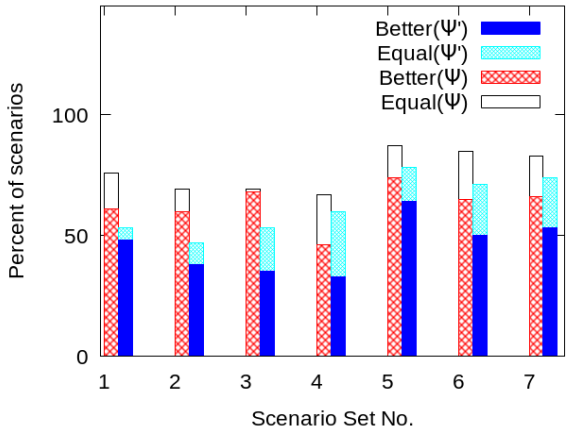


FIGURE 13. Performance of RECCE for scenarios unseen during training.

(average over 100 runs) of RECCE for scenarios in sets Ψ and Ψ' compared to the best baseline heuristic. Two scheduling strategies are deemed to perform equally if they (with a difference of $\pm 5\%$) —

- deliver the same percent of packets within the deadline, and
- have the same average delay.

The percentages are calculated relative to the number of packets generated.

A scheduling strategy is considered better than the other if it —

- delivers a larger percent of packets (by at-least more than 5%) within the deadline OR
- delivers the same percent of packets (with a difference of $\pm 5\%$) within the deadline but has an average delay of less than 90% compared to the other strategy.

Ideally, the RL agent should be better than or equal to the best baseline heuristic in 100% of the cases, as the scheduling actions are based on the baseline heuristics. However, due to the approximation introduced by deep RL for the generalized policy, RECCE may not reach this goal, as discussed in Section VI. It can be seen that RECCE gives better or equal performance in 67% to 87% of the scenarios (Figure 13, scenario set nos. 4 and 5 respectively) for scenarios from Ψ. The advantage of a generalized policy is that RECCE performs reasonably well for scenarios unseen during training as well (from 47% for scenario set no. 2 to 78% for scenario set no. 5).

E. EFFECT OF THE NUMBER OF NODES (N)

Typical industrial deployments of sensor and actuator networks tend to consist of small networks each with less than 100 nodes and a gateway, as it increases reliability and real-time performance [5]. To evaluate the scalability of RECCE, we considered three different network sizes ($N = 10$, 20 and 50). Figures 5, 11 and 15 show the CDF of average packet delay for these network sizes with a PDR of 0.7 to 1.0. In all three cases, RECCE is better than the best baseline heuristic. From Figures 6, 12 and 16, it can also be seen that RECCE is better in terms of the percentage of missed packets and the CDF of their delay.

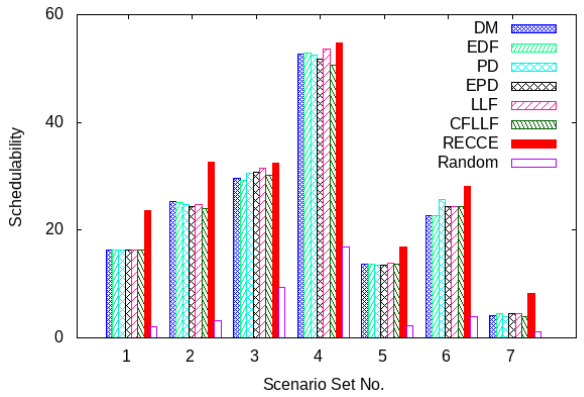


FIGURE 14. Schedulability for different scenario sets.

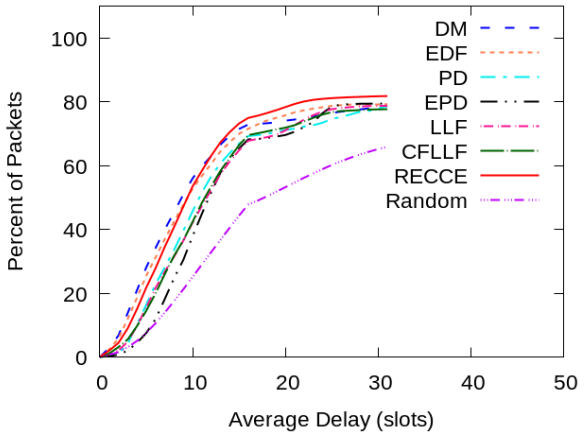


FIGURE 15. CDF of avg. Packet delay (N = 50 (Scenario set no.7), PDR from 0.7 to 1.0).

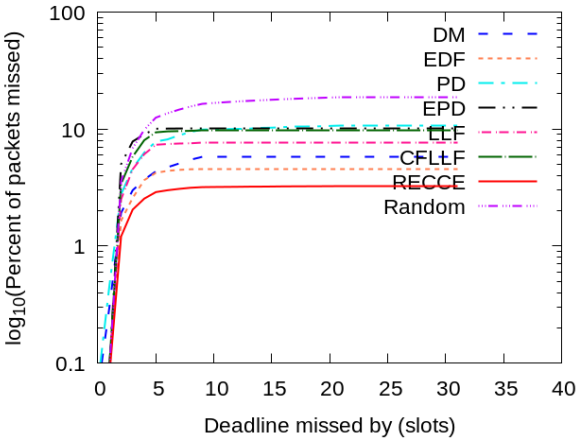


FIGURE 16. CDF of packet delay for missed packets (N = 50 (Scenario set no. 7), PDR from 0.7 to 1.0).

F. SCHEDULABILITY WITH RECCE

While RECCE addresses the problem of minimizing the missed packets, a broader goal of scheduling is generally to improve the schedulability. Please note that we consider packet loss on the links when reporting schedulability, as discussed in Section VII-A. Figure 14 depicts RECCE's performance with this metric. It can be seen that the schedulability decreases as the number of nodes increases for all scheduling policies. This is because we consider packet loss while measuring schedulability and the routes are longer in larger networks, increasing the probability of packets getting lost along the path.

It can also be seen that RECCE results in greater schedulability (up to 30% compared to the best baseline) than all the baseline scenarios, for all scenario sets. The reason for this is that RECCE reduces the number of packets missing the deadline, so that a larger percent of scenarios become feasible.

VIII. CONCLUSION

In this paper, we presented RECCE, a joint routing and scheduling scheme using deep reinforcement learning for time-constrained wireless networks. RECCE creates a centralized routing and scheduling policy with the goal of reducing the number of packets missing their deadlines and their delay overshoot. To this end, RECCE trains a DRL agent on a set of frequently-seen network scenarios to choose from popular baseline scheduling criteria and the top γ routes at each TDMA time slot. Simulation results show that RECCE performs well in terms of packets missing their deadlines, reducing up to 55% missed packets and increasing schedulability by up to 30%, relative to the best baseline heuristic. When individual scenarios are considered, RECCE can deal well with dynamic network conditions, as it performs better than the best baseline heuristic in up to 74% of the scenarios in the training set and in up to 64% of scenarios *not* in the training set.

REFERENCES

[1] A. N. Kim, F. Hekland, S. Petersen, and P. Doyle, "When HART goes wireless: Understanding and implementing the WirelessHART standard," in *Proc. IEEE Int. Conf. Emerg. Technol. Factory Autom.*, Sep. 2008, pp. 899–907.

[2] D. De Guglielmo, S. Brienza, and G. Anastasi, "IEEE 802.15.4e: A survey," *Comput. Commun.*, vol. 88, pp. 1–24, Aug. 2016.

[3] S. C. Ergen and P. Varaiya, "Tdma scheduling algorithms for wireless sensor networks," *Wireless Netw.*, vol. 16, no. 4, pp. 985–997, Jan. 2010, doi: 10.1007/s11276-009-0183-0.

[4] S. Chilukuri and A. Sahoo, "Delay-aware TDMA scheduling for multi-hop wireless networks," in *Proc. Int. Conf. Distrib. Comput. Netw.*, New York, NY, USA, Jan. 2015, pp. 1–10, doi: 10.1145/2684464.2684493.

[5] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Real-time scheduling for WirelessHART networks," in *Proc. 31st IEEE Real-Time Syst. Symp.*, Nov. 2010, pp. 150–159.

[6] X. Jin, A. Saifullah, C. Lu, and P. Zeng, "Real-time scheduling for event-triggered and time-triggered flows in industrial wireless sensor-actuator networks," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2019, pp. 1684–1692.

[7] S. Chilukuri, G. Piao, D. Lugones, and D. Pesch, "Deadline-aware TDMA scheduling for multihop networks using reinforcement learning," in *Proc. IFIP Netw. Conf.*, Jun. 2021, pp. 1–9.

[8] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[9] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM Special Interest Group Data Commun.*, New York, NY, USA, Aug. 2019, pp. 270–288, doi: 10.1145/3341302.3342080.

[10] P. Marbach, O. Mihatsch, and J. Tsitsiklis, "Call admission control and routing in integrated services networks using reinforcement learning," in *Proc. 37th IEEE Conf. Decis. Control*, vol. 1, Dec. 1998, pp. 563–568.
[11] W. Naruephiphat and W. Usaha, "Balancing tradeoffs for energy-efficient routing in MANETs based on reinforcement learning," in *Proc. IEEE Veh. Technol. Conf.*, May 2008, pp. 2361–2365.
[12] M. Gupta, A. Rao, E. Visotsky, A. Ghosh, and J. G. Andrews, "Learning link schedules in self-backhauled millimeter wave cellular networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 12, pp. 8024–8038, Dec. 2020.
[13] T. Zhang, S. Shen, S. Mao, and G.-K. Chang, "Delay-aware cellular traffic scheduling with deep reinforcement learning," in *Proc. IEEE Global Commun. Conf.*, Dec. 2020, pp. 766–774.
[14] D. Ghosal, S. Shukla, A. Sim, A. V. Thakur, and K. Wu, "A reinforcement learning based network scheduler for deadline-driven data transfers," in *Proc. IEEE Global Commun. Conf.*, Dec. 2019, pp. 253–261.
[15] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," *IEEE Access*, vol. 7, pp. 55916–55950, 2019.
[16] G. N. Seetanadi, K.-E. Årzén, and M. Maggio, "Adaptive routing with guaranteed delay bounds using safe reinforcement learning," in *Proc. 28th Int. Conf. Real-Time Netw. Syst.*, New York, NY, USA, Jun. 2020, pp. 149–160, doi: 10.1145/3394810.3394815.
[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*. [Online]. Available: http://arxiv.org/abs/1707.06347
[18] *Clip—Wolfram Language Documentation*. Accessed: Aug. 16, 2021. [Online]. Available: https://reference.wolfram.com/language/ref/Clip.html
[19] J. Y. Yen, "Finding the K shortest loopless paths in a network," *Manage. Sci.*, vol. 17, pp. 712–716, Jul. 1971. [Online]. Available: http://www.jstor.org/stable/2629312
[20] J. W. S. Liu, *Real-Time System*. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.
[21] O. Iacoboaiea, J. Krolikowski, Z. Ben Houidi, and D. Rossi, "Real-time channel management in WLANs: Deep reinforcement learning versus heuristics," in *Proc. IFIP Netw. Conf.*, 2021, pp. 1–9.
[22] Openai. *GYM: A Toolkit for Developing Comparing Reinforcement Learning Algorithms*. Accessed: Dec. 2020. [Online]. Available: https://gym.openai.com/
[23] I. Kostrikov. (2018). *Pytorch Implementations of Reinforcement Learning Algorithms*. [Online]. Available: https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail



SHANTI CHILUKURI (Member, IEEE) received the Bachelor of Engineering degree in electrical and electronics engineering and the Master of Technology degree in computer science and technology from the Andhra University College of Engineering, Visakhapatnam, India, and the Ph.D. degree from the Indian Institute of Bombay, in 2012. She has nearly 17 years of experience teaching undergraduate and post graduate students of computer science and engineering. Her research interests include resource allocation in wireless networks and application of machine learning for constrained networks.



DIRK PESCH (Senior Member, IEEE) received the Dipl.Ing. (M.Sc.) degree in electrical and electronic engineering from RWTH Aachen University, Germany, and the Ph.D. degree in electrical and electronic engineering from the University of Strathclyde, Glasgow, Scotland. He is currently a Professor of computer science with University College Cork, Ireland. Prior to joining UCC, he was a Professor and the Head of the Nimbus Research Centre, Cork Institute of Technology (now Munster Technological University). His research interests include architecture, design, algorithms, and performance evaluation of low power, dense and moving wireless/mobile networks and services for the Internet of Things, and cyber-physical system's applications and interoperability issues associated with IoT applications. He is a Principle Investigator at the Science Foundation Ireland funded initiatives, such as the CONNECT Centre for Future Networks and the CONFIRM Centre for Smart Manufacturing. He is the Director of the SFI Centre for Research Training in Advanced Networks for Sustainable Societies. He is on the editorial board for a number of international journals and contributes to international conference organization in his area of expertise.