Deadline-Aware TDMA Scheduling for Multihop Networks Using Reinforcement Learning

Shanti Chilukuri
School of Computer Science and IT,
University College Cork, T12 K8AF Cork, Ireland
s.chilukuri@cs.ucc.ie

Guangyuan Piao
Department of Computer Science, Hamilton Institute,
Maynooth University, Maynooth, Co Kildare, Ireland
guangyuan.piao@mu.ie

Diego Lugones
Nokia Bell Labs,
Dublin, Ireland
diego.lugones@nokia-bell-labs.com

Dirk Pesch
School of Computer Science and IT,
University College Cork, T12 K8AF Cork, Ireland
d.pesch@cs.ucc.ie

***Abstract*—Time division multiple access (TDMA) is the medium access control strategy of choice for multihop networks with deterministic delay guarantee requirements. As such, many Internet of Things applications use protocols based on time division multiple access. Optimal slot assignment in such networks is NP-hard when there are strict deadline requirements and is generally done using heuristics that give sub-optimal transmission schedules in linear time. However, existing heuristics make a scheduling decision at each time slot based on the same criterion without considering its effect on subsequent network states or scheduling actions. Here, we first identify a set of node features that capture the information necessary for network state representation to aid building schedules using Reinforcement Learning (RL). We then propose three different centralized approaches to RL-based TDMA scheduling that vary in training and network representation methods. Using RL allows applying diverse criteria at different time slots while considering the effect of a scheduling action on meeting the scheduling objective for the entire TDMA frame, resulting in better schedules. We compare the three proposed schemes in terms of how well they meet the scheduling objectives and their applicability to networks with memory and time constraints. One of the schemes proposed is RLSchedule, which is particularly suited to constrained networks. Simulation results for a variety of network scenarios show that RLSchedule reduces the percentage of packets missing deadlines by up to 60% compared to the best available baseline heuristic.**

I. Introduction

The availability of low-cost and low-form factor sensors and actuators in recent times has resulted in their usage in a wide variety of applications such as surveillance, pollution monitoring, energy monitoring and control and predictive maintenance and automation of factories. Some of these applications have strict Quality of Service (QoS) requirements – especially those of low delay, high reliability and throughput. In such applications, the generated data is sent as packets which are associated with a deadline and packets missing their deadlines can lead to catastrophic results. In light of the strict QoS requirements combined with the resource-constrained nature of the devices and harsh network conditions, design and operation of the network for such applications is a challenging task.

One major contributor to end-to-end delay is the delay encountered at the medium access control (MAC) layer. MAC protocols for real-time data transfer (such as WirelessHART [1] and IEEE 802.15.4-TSCH [2]) are based on a slot assignment strategy with time division multiple access (TDMA), which can yield deterministic delay guarantees. In most such cases, the slot assignment is done by a central network controller, which has knowledge of the network conditions and builds a schedule (slot and channel assignment in multi-channel networks) to be followed by the nodes. Optimal slot scheduling is NP-hard [3] and hence, several heuristics have been proposed to find a (generally sub-optimal) schedule with low computation time. Such heuristics reduce the search space by focusing on a (single) criterion (e.g., minimum laxity, minimum proportional deadline) for scheduling at any time slot, with a myopic view based on the state of packet queues at that instant.

In this paper, we propose RLSchedule, a framework that uses Reinforcement Learning (RL) for TDMA slot scheduling in networks with strict time constraints. In RL, an optimal policy is learned by taking an action (based on the system's current state) from a set of actions at each step and observing a reward in return [4]. The optimal policy is one that maximizes the overall long-term reward gained. Using RL for scheduling allows the scheduler to apply diverse criteria for scheduling at each time slot. In addition, the scheduler can also consider the effect of an action on the subsequent states that the network enters and possible actions therein (within a TDMA frame). RLSchedule's state, action and reward functions are crafted carefully, so that the scheduler includes schedules that better meet the scheduling objective

This work has received funding in part, from the European Union's Horizon 2020 Research and Innovation Programme under the EDGE COFUND Marie Sklodowska Curie grant agreement No. 713567 and from the Science Foundation Ireland under CONNECT Centre grant no. 13/RC/2077 and Enable Centre grant no. 16/SP/3804.

in its search space and finds the best schedule among them.

For networks with strict deadline requirements, a network scenario (the network graph together with the packet deadline, periodicity and route information) is said to be *schedulable* if a schedule where all packets meet their deadlines can be found. Such a schedule is called a *feasible schedule*. However, depending on the network topology, there may not exist a schedule where all packets meet their deadline. The goal of scheduling for networks with strict time delay requirements should in fact be *to find a schedule where the least possible number of packets miss the deadline by the least amount of time.* This provides a much finer basis for checking if a schedule meets the application delay requirements and hence for comparing different scheduling heuristics. In view of this, the major contributions of this paper are:

- identification of a set of node features that help in state representation for RL-based scheduling.
- proposal of three RL-based schemes for centralized TDMA scheduling in networks with strict packet deadlines. These schemes vary in the way the RL agent is trained and in the way the network state is represented.
- study of the suitability of each of these schemes for building TDMA schedules in such networks in terms of how well they meet the scheduling goal for previously seen or unseen network scenarios, the time taken for building the schedule and the memory required at the network controller to store the built model(s).
- evaluation of the performance of the scheme most suitable to constrained networks (called RLSchedule) for different number of network nodes, deadline requirements and channels.

Simulation results show that RLSchedule achieves the scheduling objective better compared to baseline and popular heuristics in the literature. In case multiple schedules are possible for a particular scenario with no missed packets, RLSchedule finds a schedule with lower packet delay than existing heuristics. Otherwise, it finds a schedule where fewer number (by up to 60%) of packets miss their deadline by a smaller time margin compared to existing heuristics.

II. Related Work

TDMA scheduling heuristics for networks requiring guaranteed delay was studied previously. Most of them [5]–[7] propose heuristics with the goal of minimizing the TDMA frame length with slot-reuse. However, standards such as WirelessHART do not allow concurrent transmissions in the same channel in order to avoid interference. Also, most of the past work has focused on convergecast. In this paper, we focus on the more general problem of any-to-any communication requiring strict delay guarantee without slot reuse.

The closest to our network model is the work in [3], where the authors propose the Conflict-Free Least Laxity First (CFLLF) heuristic that improves schedulability of a set of network scenarios compared to other baseline heuristics. For applications requiring strict guarantees but with network scenarios which are not schedulable, reducing the number of packets missing the deadline and the time by which they miss the deadline is a finer and important goal to achieve than that in [3]. Our proposed scheme achieves this goal much better than CFLLF, without adversely affecting the schedulability.

Reinforcement Learning (RL) has shown considerable promise in learning better resource allocation policies (e.g., [8]–[11]), especially for job and task scheduling. In [12], the authors propose an RL-based scheme for finding near-optimal task schedules. More recently, [8] proposes Decima for job scheduling in data processing clusters. Optimal traffic scheduling in cellular networks using RL was studied in [10]. In [13], the authors propose an RL-based scheduler for scheduling flows in a software-defined network to schedule the pacing rate of the sources. Though job scheduling and traffic scheduling are basically resource allocation problems much like slot allocation, the state space, action space and reward design is different for these problems. In [9], the authors propose an RL-based scheduling policy for link scheduling in backhaul networks to minimize the packet delay. Though this is also slot scheduling, RLSchedule considers a more general network model and has a scheduling objective based on packet deadlines rather than the average delay. To the best of our knowledge, RL has not been explored for TDMA slot scheduling in networks with deadlines.

In [14], the authors propose algorithms to reserve slots for event-based flows without adversely affecting the performance of time-based flows. One of these schemes is the virtual period (VP) method where the actual scheduling is done using any standard heuristic such as earliest deadline first. In this paper, we consider a mixture of event and time-based flows to be scheduled in each hyper-frame such that the scheduling objective is met for both types of flows. As such, RLSchedule can be used instead of the standard heuristic together with the VP method of [14].

III. Network Model

In this section, we explain the network model (notation in Table I) we consider for scheduling. For a particular network scenario, we consider a network with N number of devices (network nodes) that transmit data in any one of the available M channels. Each of the devices can be the source or destination of data. The nodes transmit data in packets using wireless links, with each link characterised by its packet loss probability.

Each *flow* of data consists of a packet that travels from source to destination along a route and the set of flows for the scenario is denoted by F. Flows may be of two types – time-triggered (periodic) or event-triggered (aperiodic). Each flow f_i is associated with

1) a start time slot s_i when the packet is generated at the source node,
2) a relative deadline $reld_i$, which is the number of slots from s_i before which the packet has to reach the destination,

TABLE I
NOTATION

Symbol	Meaning
N	Number of nodes in the network
M	Number of channels
F	set of flows in the network
s_i	start time slot of the i^{th} flow
$reld_i$	relative deadline of the i^{th} flow
w_i	priority of the i^{th} flow
p_i	period of the i^{th} flow
$\vec{R}_i$	optimal route of the i^{th} flow
$absd_i$	absolute deadline of the i^{th} flow
β	hyper-period of the network scenario
γ_i	a transmission of a packet of the i^{th} flow
u_{γ_i}	no. of hops from source to transmission γ_i
v_{γ_i}	no. of hops from transmission γ_i to the destination
tr_{γ_i}	remaining time of transmission γ_i at slot t
Γ_k^t	set of transmissions at node k at time slot t
$\mathcal{T}^t$	set of transmissions in the network at time slot t
Ψ	set of most frequently seen network scenarios
α	ratio of deadline to period of a flow
ρ_{min}	minimum exponent of period of a flow
ρ_{max}	maximum exponent of period of a flow
$(x1_k^t, x2_k^t, x3_k^t, x4_k^t)$	features of node k in slot t
g^t	aggregated graph feature in slot t
$(ne1_k^t, ne2_k^t)$	2D embedding of node k in slot t
ge^t	aggregated graph embedding in slot t

3) a priority w_i based on the application generating the flow,
4) a period p_i if the flow is periodic, and
5) a route $\vec{R}_i$ that the packet takes as it flows to the destination. The route taken by a packet is determined by a standard routing algorithm based on Dijkstra's, such that the route has minimum cumulative packet loss probability. The number of hops from the source to the destination along the route is denoted by h_i.

The *absolute deadline* of a flow is the slot by which the packet has to reach its destination and is denoted by $absd_i$, with $absd_i = reld_i + s_i$.

In centralized scheduling, the network controller has above information about all flows, in addition to the current adjacency matrix for the network graph ([15] presents a way of facilitating this efficiently). It creates a TDMA schedule assigning slots and channels to nodes and sends this to nodes. It is sufficient if a schedule is found for hyper-period β, where β is the least common multiple of all periods. A periodic flow f_i has a new packet generated at every $(s_i + k * p_i)^{th}$ slot in a hyper-period, where $k \in [0, \beta \mod p_i]$.

All nodes can transmit/receive as per the received schedule until a new schedule is received. A packet generated reaches its destination after a series of *transmissions*, where each transmission (a packet at a particular hop) belonging to the i^{th} flow is denoted by γ_i. A transmission γ_i is characterized by the following —

- u_{γ_i}, which is the number of hops from the source to the destination,
- v_{γ_i}, which is the number of hops from γ_i to the destination and
- tr_{γ_i}, the remaining time of a transmission γ_i at time slot t is $tr_{\gamma_i} = reld_i - (t - s_i)$

At any given time slot t, the set of transmissions possible at node k is denoted by Γ_k^t and the set of transmissions across the entire network is $\cup_{i=1}^{N} \Gamma_i^t$. Two transmissions are non-conflicting if they do not have a common source or destination. In the absence of channel reuse across the network (as is common with standard protocols like WirelessHART [1] and IEEE 802.15.4-TSCH [2]), non-conflicting transmissions can be assigned different channels in the same slot. Out of the $\sum_{i=1}^{N} |\Gamma_i^t|$ transmissions at slot t, the scheduling algorithm chooses ("schedules") a maximum of M non-conflicting transmissions to be advanced to their next hop nodes. The goal of scheduling is to assign slots and channels to nodes for the hyper-period β in such a way that all flows reach their destinations before their deadlines. Such a schedule is called a *feasible schedule*. However for some network scenarios, it may not be possible to find a feasible schedule, in which case the goal is to minimize the number of packets missing the deadline and by how much they miss the deadline.

A. The Optimal Scheduling Policy Baseline

As one of the baseline scheduling strategies, we consider an *optimal* scheduling policy that is somewhat similar to that in [3]. As per this policy, at each time slot t, all possible subschedules of the transmissions in set $\cup_{i=1}^{N} \Gamma_i^t$ are considered for the branch and bound scheduling tree. To reduce complexity, those subschedules where packets miss their deadline (i.e., transmissions with laxity < 0) are not considered for branching, as in [3]. However, while the optimal algorithm proposed in [3] exits once *any* feasible schedule is found, our version of the optimal algorithm exits only when a feasible schedule with the minimum possible delay is found. This increases the complexity, but provides a basis to check if RLSchedule can indeed find schedules that are closer to optimal compared to other baseline heuristics in the case of feasible schedules.

IV. RLSCHEDULE PRINCIPLES

RLSchedule uses Proximal Policy Optimization (PPO [16]), a policy gradient method in RL for learning the policy of the scheduler. During training, the RL agent steps through the environment by taking a random action from the action space and observes the instantaneous reward. After training for enough episodes, the RL agent can converge to an optimal policy such that the action taken at any state leads to maximum cumulative reward for that episode. More details on policy gradient methods and RL are in [4]. At each step, a set of transmissions is chosen to be scheduled in a slot. Training runs in episodes, where each episode starts with slot 0 of the hyper-period and ends at slot $\beta - 1$.

A. State Space

To learn an efficient scheduling policy, it is necessary to represent the network state in a way to take into consideration the factors that affect the goal of scheduling. Let Γ_k^t be the set of transmissions at node k at the t^{th} time slot. We consider four node features $x1_k^t$ to $x4_k^t$ which are:

- the number of transmissions at that node, $x1_k^t = |\Gamma_k^t|$
- the minimum remaining time of a transmission queued at the node. Thus, $x2_k^t = \min_{\forall \gamma_i \in \Gamma_k^t}(tr_{\gamma_i})$
- the maximum number of remaining hops of a transmission at the node, $x3_k^t = \max_{\forall \gamma_i \in \Gamma_k^t}(v_{\gamma_i})$
- the minimum ratio of remaining time and remaining hops of a transmission at the node. Thus, $x4_k^t = \min_{\forall \gamma_i \in \Gamma_k^t}(\frac{tr_{\gamma_i}}{v_{\gamma_i}})$

In addition to the above features, we also consider a simple aggregated graph feature g^t (average of the features of all the nodes). The state of the network at any t is hence a vector of dimension $4N$+1.

B. Action Space

The total set of transmissions to be scheduled at slot t is $\mathcal{T}^t$, where $\mathcal{T}^t = \bigcup_{i=1}^{N} \Gamma_i^t$. Considering all the possible subsets of $\mathcal{T}^t$ as possible actions results in a very large action space. To reduce the action space size, we consider six possible actions at any slot to assign the M available channels. The first five of them correspond to choosing the first M non-conflicting transmissions from $\mathcal{T}^t$ based on a proven baseline heuristic. The baseline heuristics considered are listed below, together with their scheduling strategy.

- Deadline Monotonic (DM) – transmissions with least relative deadline of their flow are scheduled first.
- Earliest Deadline First (EDF) – transmissions with the least absolute deadline are scheduled first.
- Proportional Deadline (PD) – transmissions with least value of relative deadline divided by the total number of hops for that flow are scheduled first.
- Earliest Proportional Deadline (EPD) – transmissions with the least remaining time divided by the remaining number of hops are scheduled first and
- Least Laxity First (LLF) – transmissions with the least laxity i.e., remaining time minus the remaining number of hops are scheduled first.

In addition to the above five actions, RLSchedule considers a sixth possible action that sorts the nodes in the ascending order of their features and chooses the top M non-conflicting transmissions with the minimum set of features. In case multiple transmissions satisfy the scheduling criteria (e.g., have the same least relative deadline), transmissions with highest priority w are chosen for scheduling.

C. Reward Design

At each step of learning, the RL agent gathers a reward which is inversely proportional to the delay incurred by all the packets that reach their destination in that slot. In addition, a large negative reward is gathered for each packet that reaches the destination in that slot, but has missed its deadline. This reward design minimizes the number of packets missing their deadlines. Between two schedules that have the same number of missed packets, the schedule with lower average packet delay has a better total reward.

D. Building a Scheduling Policy

The RL agent with the above state, action and reward design is trained over multiple episodes. Each episode starts with slot 0 of the hyper period and ends at the last slot of the hyper period. In centralized scheduling protocols such WirelessHART and TSCH, the network controller has to build a new schedule whenever the network scenario (the adjacency matrix, number of nodes, flows and channels and flow parameters) changes. Dynamic network conditions because of the wireless links, node mobility and presence of event-based flows may necessitate frequent changes of schedules. Also, the schedule needs to be built in as little time as possible. RL generally suffers from poor convergence time for training. To overcome this problem, we propose building an RL policy model offline (e.g., at a server) with final policy model/s stored at the controller to build schedules by inference from the model. We consider three (most-intuitive) schemes to train the agent.

Algorithm 1 Scheme1(ψ)

Input: Number of nodes N, Number of channels M, Network adjacency graph A, Flow information F for one scenario ψ
Output: Custom Policy model θ for scenario ψ

```
j = 0,  t = 0
while j < max_updates do
    while k < batch_size do
        Calculate node features (x1_k^t, x2_k^t, x3_k^t, x4_k^t) for k ∈ [1, N] and g^t
        Perform action and calculate reward as per PPO
        Increment k, t
        if t=hyper-period of scenario ψ then
            t = 0
        end if
    end while
    Update policy θ as per PPO
    Increment j
end while
```

- *Scheme 1*: The agent is trained with the same network scenario in each episode, using the raw node features $(x1_k^t, x2_k^t, x3_k^t, x4_k^t)$ and the aggregated graph feature g^t as the input at each step (Algorithm 1). At the end of training, a policy model is built for scheduling this network scenario. If Ψ is the set of frequently seen topologies by the network controller, it stores a separate model for each scenario in Ψ.
- *Scheme 2*: The agent is trained with a random network scenario from set Ψ (most frequently seen scenarios) in each episode, using raw node features $(x1_k^t, x2_k^t, x3_k^t, x4_k^t)$ and aggregated graph feature g^t as input at each step (Algorithm 2). At the end of training, a single, generalized model is built for the entire set

Algorithm 2 Scheme2(Ψ)

Input: Number of nodes N, Number of channels M, Network adjacency graph A and Flow information F for each scenario in Ψ
Output: Generalized Policy model θ for the set Ψ

$j = 0, \quad t = 0$
Choose random scenario ψ from Ψ
while $j < max\_updates$ **do**
  while $k < batch\_size$ **do**
    Calculate node features $(x1_k^t, x2_k^t, x3_k^t, x4_k^t)$ for $k \in [1, N]$ and g^t
    Perform action and calculate reward as per PPO
    Increment k, t
    if t=hyper-period of scenario ψ **then**
      $t = 0$
      Choose a random scenario ψ from Ψ
    end if
  end while
  Update policy θ as per PPO
  Increment j
end while

Algorithm 3 Scheme3(Ψ)

Input: Number of nodes N, Number of channels M, Network adjacency graph A and Flow information F for each scenario in Ψ
Output: Generalized Policy model θ for the set Ψ

$j = 0, \quad t = 0$
Choose random scenario ψ from Ψ
while $j < max\_updates$ **do**
  while $k < batch\_size$ **do**
    Calculate node features $(x1_k^t, x2_k^t, x3_k^t, x4_k^t)$ for $k \in [1, N]$
    Calculate node embeddings $ne1_k^t, ne2_k^t$ for $k \in [1, N]$ and aggregated graph embedding ge^t
    With embeddings as the state, perform action and calculate reward as per PPO
    Increment k, t
    if t=hyper-period of scenario ψ **then**
      $t = 0$
      Choose a random scenario ψ from Ψ
    end if
  end while
  Update policy θ as per PPO
  Increment j
end while

of topologies in Ψ. In case the controller frequently sees scenarios not in the training set resulting in poor scheduling performance, the information about this new set of scenarios is sent to the server, which retrains the RL agent with the new set of frequently seen topologies.

- *Scheme 3*: The agent is trained with a random network scenario from set Ψ in each episode, using a 2 dimensional node embedding $(ne1_k^t, ne2_k^t)$ and the aggregated graph embedding ge^t built from the raw node features as input (Algorithm 3). The embeddings are built using graph representation learning (GraphSage [17]). At the end of training, a single, generalized model is built as in Scheme 2 and the model is rebuilt periodically if necessary.

The average time taken to find the schedule using each of these schemes, the optimal scheduling policy and a baseline heuristic (DM) is given in Figure 1. The graph is in log scale for the time (in milliseconds) taken for each of these scheduling strategies. Only one baseline heuristic is shown as there is not much difference in the time taken by different baseline heuristics. The optimal policy is shown only for a 10 node network, as the time taken increases exponentially with the number of nodes, flows or channels. As seen among Schemes 1, 2 and 3, Scheme 3 takes much more time for building a schedule by inference from the model (and also for building the model) compared to Schemes 1 and 2, without much improvement in the scheduling goal achievement. Since the schedule in most networks with time constraints needs to be built very quickly, the time taken to build a schedule needs to be as small as possible. Though the time taken by Schemes 1 and 2 is more than that taken by the baseline heuristic, it does not increase exponentially with the number of nodes and gives much better results than the baseline heuristics. Hence, we consider the choice between Schemes 1 and 2 in the discussion below.

We compare these two schemes in terms of their memory requirement at the network controller, efficiency in meeting the scheduling goal, training cost and efficiency in meeting

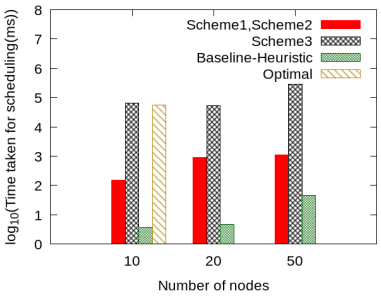

Fig. 1. Time taken for building a schedule (excluding training)

the scheduling goal for topologies not considered for training.

- Memory requirement at the network controller: For Scheme 1, if B bytes is the memory required to store a policy model and the network controller needs to store $|\Psi|$ models, one for each network scenario it sees, $B|\Psi|$ bytes of memory are required. For Scheme 2, since a single model is used for scenarios in Ψ, the memory required is B bytes. Clearly, Scheme 2 is better for memory constrained network controllers.
- Efficiency in meeting the scheduling goal: Figure 2 compares the performance of Schemes 1 and 2 with that of the best baseline heuristic in terms of the average packet delay, for twenty different network scenarios. It can be seen that both Schemes 1 and 2 have a delay less than or equal to that of the best heuristic in all cases. This shows that the RL agent can learn a scheduling policy that is better than or at least as good as the best heuristic. Further, it can be seen that Scheme 1 has a delay that is lower than or equal to the delay by Scheme 2 in all cases. Thus, Scheme 1 is better than Scheme 2 in terms of meeting the scheduling goal.
- Training cost: Scheme 1 builds $|\Psi|$ models while Scheme 2 builds a single model. However, building a general model requires more episodes of training than

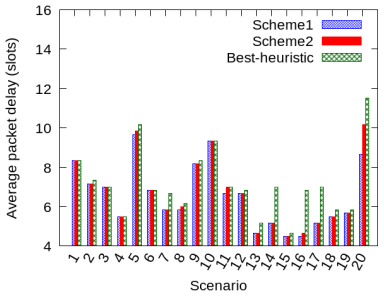

Fig. 2. Efficiency in meeting the scheduling goal (20 nodes, 6 flows)

building a custom model for one scenario. In any case, since RLSchedule performs offline training (possibly at a server), we do not consider training cost to be the deciding factor between Schemes 1 and 2.

- Efficiency in meeting the scheduling goal for topologies not considered for training: When the network controller has to build a schedule for a network scenario that is not in Ψ, Scheme 2 is better as it has a generalized model. Scheme 1, which has a custom model for each scenario in Ψ, may not be able build a good schedule for such unseen scenarios.

Based on the above discussion, Scheme 1 is the best when the network controller is not memory-constrained and does not encounter new network scenarios frequently. Since we consider IoT applications, the network controllers generally do not have a lot of memory. Also, node mobility, changing channel conditions and event-based flows result in a variety of network scenarios. Hence, for the rest of this paper, we focus on Scheme 2 (RLSchedule).

E. RLSchedule in practice

Since the resources (time and computing power) taken for learning the policy may not be available at the network controller in real networks, RLSchedule uses an offline learning scheme. As per this scheme, the network controller gathers the information about the Ψ most frequently seen network scenarios and sends this information to a server. The server runs the RL training module and sends the resulting scheduling policy to the network controller. The scheduler uses this policy to find schedules for any scenario it sees.

It may be possible that the controller sees topologies that have not been included in the training set Ψ. If the current policy gives poor results for such unseen topologies, the network controller sends them to the server and they are added to the training set. This periodic refining of the model based on the most frequently seen topologies helps improve the scheduling performance.

V. Simulation Setup and Results

In this section, we discuss the network scenario generation and present the results of simulation.

A. Network Scenario Generation

Network scenarios were generated each with N nodes (spread randomly in a 100mx100m area) and f flows, starting and ending at random nodes. Links between the nodes were generated with random packet loss probabilities. The route for each flow was determined using Dijkstra's shortest path algorithm. The route with the minimum cumulative packet loss probability is considered the best. The period p of each flow is harmonic [18] and was generated randomly between $2^{\rho_{min}}$ and $2^{\rho_{max}}$ with the deadline being $p * \alpha$, where $\alpha \in (0, 1]$ is the ratio between the deadline and the period of a flow. Based on the discussion in Section IV-D, we consider a generalized model (Scheme 2) for a set of Ψ (=250) most frequent scenarios for each set of results while presenting the results of RLSchedule. The results are presented for this entire set of scenarios. We consider five different sets of scenarios with different parameters shown in Table II.

TABLE II
Scenarios considered for Simulation

Scenario set no.	N	M	|F|	ρ_{min}	ρ_{max}	α
1	10	2	4	4	4	0.75
2	10	1	4	4	4	0.75
3	20	2	6	5	5	0.75
4	50	8	15	5	6	0.5
5	20	2	6	4	4	0.75

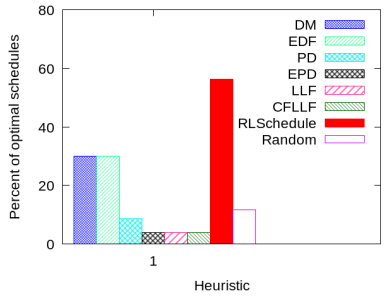

Fig. 3. Percent of cases in which an optimal schedule is found (10 nodes, 4 flows, 2 channels, Scenario set no.1)

For evaluating the baseline heuristics and the optimal policy, we use a custom simulator built by us, while for implementing RLSchedule we use OpenAI's gym toolkit [19], building on the PPO implementation in PyTorch provided by [20], using the same packet-level abstraction for both. The RL agent was trained for 300000 to 500000 episodes in each case, with an entropy loss coefficient of 0.025 to ensure sufficient exploration. The models generated occupy a maximum space of 3MB each.

B. Comparision with the Optimal Policy

We first consider network scenarios with 10 nodes, 4 flows and two channels. The period of the flows is taken to be 16

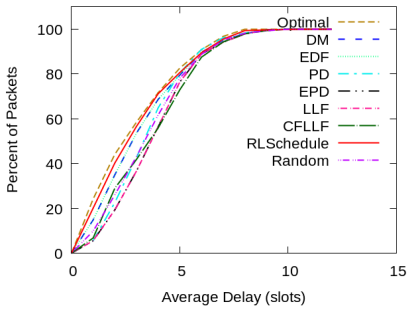

Fig. 4. CDF of Avg. Packet Delay (10 nodes, 4 flows, 2 channels, Scenario set no.1)

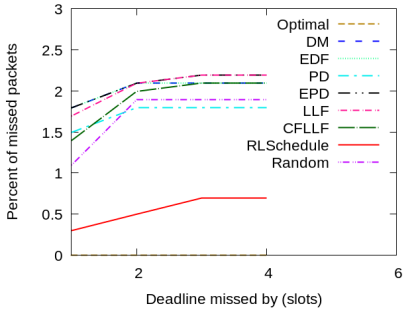

Fig. 5. CDF of Packet Delay for Missed Packets (10 nodes, 4 flows, 2 channels, Scenario set no.1)

slots ($\rho_{min} = \rho_{max} = 4$) and the deadline to period ratio $\alpha = 0.75$. For these simple network scenarios, we compare:

- the optimal scheduling policy discussed in III-A,
- the five baseline policies discussed in Section IV-B,
- the Conflict Free Least Laxity First (CFLLF) policy proposed in [3],
- RLSchedule and
- a random policy where one of the six actions discussed in Section IV-B is taken at random at each time step.

Figure 3 depicts the percent of scenarios in which a heuristic finds an optimal schedule for the eight strategies (excluding optimal) listed above. It can be seen that RLSchedule finds the optimal schedule in a much larger number of scenarios (≈56% of total scenarios considered), while taking much less time (Figure 2) compared to the optimal strategy. It also finds better schedules than all other baseline heuristics in 39% of the scenarios considered. Thus, when trained properly, RLSchedule can perform better than or equal to the best baseline heuristic.

Figure 4 gives the CDF of packet delays for these nine scheduling policies. It can be seen that RLSchedule results in lower average packet delay than the other heuristics. Figure 5 shows the CDF of the number of timeslots by which packets miss their deadline. While the optimal policy can find a feasible schedule (i.e., no missed packets) for this small network size for all the topologies considered, the rest of the policies result in a few missed packets across the 250 scenarios considered. It can be seen that RLSchedule has the least percent of missed packets among the policies compared, except for the optimal policy.

C. Effect of number of channels

Next, we consider network scenarios with above-mentioned parameters, but with a single channel. Since the optimal policy finds a schedule only if it is feasible and most of the scenarios in this set are not, we do not plot the results of the optimal policy. As is expected, this set of scenarios has greater average packet delay and percent of packets missing the deadline for all scheduling policies. Figure 6 shows the CDF of the average packet delay and Figure 7 shows the CDF of time slots by which the deadline is missed. It can be seen that the average packet delay of RLSchedule is greater than some of the baseline heuristics, but it performs better in terms of packets missing the deadline. This is because missing a packet is penalized more than increased delay in the reward design in alignment with the goal of scheduling. *RLSchedule finds those schedules where fewer number of packets miss their deadlines and by fewer slots, though packets with more delay budget have larger delays contributing to higher overall delay.*

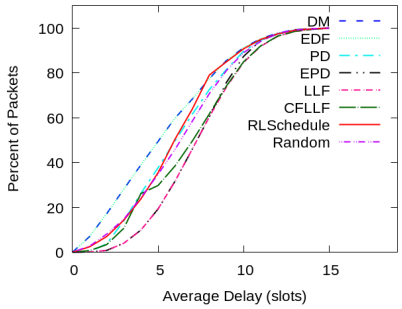

Fig. 6. CDF of Avg. Packet Delay (10 nodes, 4 flows, 1 channel, Scenario set no.2)

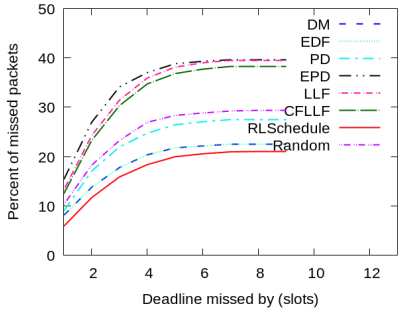

Fig. 7. CDF of Packet Delay for Missed Packets (10 nodes, 4 flows, 1 channel, Scenario set no.2)

D. Effect of network size

Figures 8 and 9 show the results for a set of network scenarios with N=20, F=6, M=2, $\rho_{min} = \rho_{max} = 5$,$\alpha = 0.75$. Again, RLSchedule performs better than all other strategies. The optimal scheduling policy takes a very long time and hence is not considered for this set of scenarios. It can be seen that the number of packets missing the deadline decreases

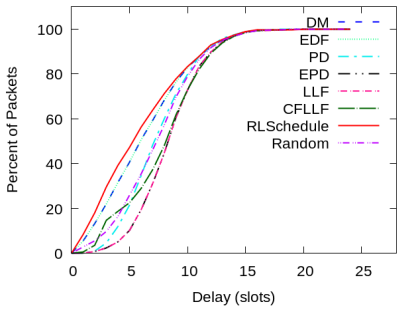

Fig. 8. CDF of Packet Delay (20 nodes, 6 flows, 2 channels, Scenario set no.3)

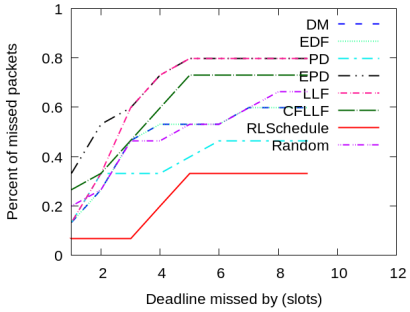

Fig. 9. CDF of Packet Delay for Missed Packets (20 nodes, 6 flows, 2 channels, Scenario set no.3)

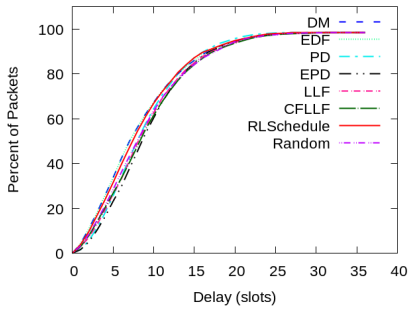

Fig. 10. CDF of Packet Delay (50 nodes, 15 flows, 8 channels, Scenario set no.4)

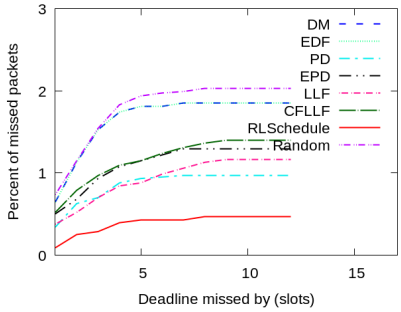

Fig. 11. CDF of Packet Delay for Missed Packets (50 nodes, 15 flows, 8 channels, Scenario set no.4)

compared to 10 node, 4 flow networks (Figure 5) with the rest of the parameters being the same, due to less scheduling conflicts. Figures 10 and 11 depict the packet delay CDF and the CDF of deadline missed by missed packets for a larger network with more number of flows (N=50, F=15, M=8, $\rho_{min} = 5$, $\rho_{max} = 6$,$\alpha = 0.5$). It can again be seen that though the average packet delay is slightly worse than the best baseline heuristic with RLSchedule, the performance in terms of missed packets is much better.

E. Effect of stricter deadlines

To study the effect of smaller deadlines on the schedules built, we consider scenarios with 20 nodes and all other parameters the same as those in SectionV-D, except for the time periods and hence, the deadlines. The results are presented in Figures 12 and 13 for such scenarios (N=20, F=6, M=2, $\rho_{min} = \rho_{max} = 4$,$\alpha = 0.75$). Compared to Figure 9, the number of packets missing their deadlines is more because of smaller deadlines, even though the rest of the parameters are the same. For this set, RLSchedule performs much better than all other schemes both in terms of the average packet delay and the deadline criterion, with the percentage improvement being much greater than that for more lenient deadlines.

F. Schedulability

While the CDF of packet deadline overshoot gives one way of comparing the scheduling strategies, a more traditional approach is to check the schedulability (percentage of total scenarios in which a feasible schedule can be found). Figure 14 shows the schedulability for each set of scenarios considered. The schedulability of the optimal strategy is shown only for a small network as it takes exponentially large time for bigger networks. It can be seen that RLSchedule results in a schedulability that is equal to or better than the best baseline heuristic for all types of scenarios.

The following are the main conclusions from the results:

- In case all the network scenarios in set Ψ have feasible schedules, RLSchedule results in packet delay CDF that is better than all baseline heuristics.
- In case some of the scenarios in Ψ do not have feasible schedules, RLSchedule has lower number (by up to 60%) of packets missing the deadline and the deadline is missed by fewer number of slots, though the CDF of packet delay may be slightly more than the best (in terms of average delay) baseline heuristic.
- This improvement can be seen for network scenarios of different sizes (nodes and flows), channels and time period (and deadline) ranges.
- Schedulability with RLSchedule is equal to or better than that of the best baseline heuristic in all cases.

VI. Conclusions

In this paper, we explored the efficacy of reinforcement learning (RL) for finding TDMA schedules in networks with strict time constraints. We identified a set of features of the packet queues that help in making the scheduling decision.

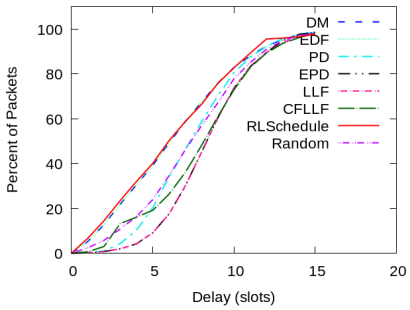

Fig. 12. CDF of Packet Delay with Stricter Deadlines (20 nodes, 6 flows, 2 channels, Scenario set no.5)

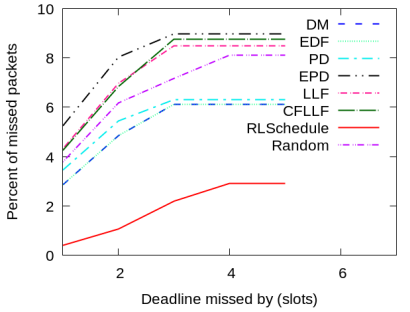

Fig. 13. CDF of Packet Delay for Missed Packets with Stricter Deadlines (20 nodes, 6 flows, 2 channels, Scenario set no.5)

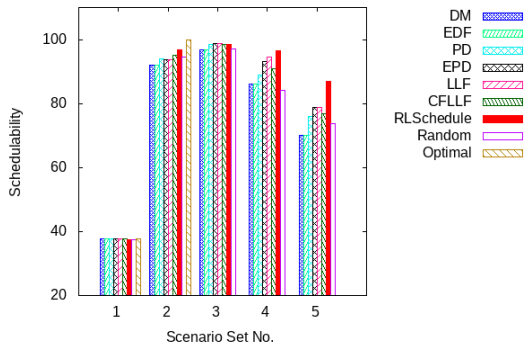

Fig. 14. Schedulability for different sets of scenarios

We then proposed and studied three different schemes of RL-based scheduling with custom or generalized models, with or without node embedding. Of these, the second scheme is most suited for networks where the scheduling decision has to be made quickly and changing network scenarios are seen frequently. We evaluated this scheme for different network sizes, channels and deadline requirements. Simulation results show that RLSchedule can result in better average packet delay than the best baseline heuristic in case a set of scenarios with feasible schedules. In case it is not possible to find a feasible schedule, RLSchedule reduces the number of packets missing the deadline and the number of slots by which the deadline is missed, even compared to the best baseline heuristic. In the future, we plan to explore the feasibility of joint route and scheduling optimization using RL.

REFERENCES

[1] A. N. Kim, F. Hekland, S. Petersen, and P. Doyle, "When HART goes wireless: Understanding and implementing the wirelesshart standard," in *2008 IEEE International Conference on Emerging Technologies and Factory Automation*. IEEE, 2008, pp. 899–907.

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

[3] A. Saifullah, Y. Xu, C. Lu, and Y. Chen, "Real-time scheduling for WirelessHART networks," in *2010 31st IEEE Real-Time Systems Symposium*, 2010, pp. 150–159.

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

[5] S. C. Ergen and P. Varaiya, "TDMA scheduling algorithms for wireless sensor networks," *Wirel. Netw.*, vol. 16, no. 4, p. 985997, May 2010. [Online]. Available: https://doi.org/10.1007/s11276-009-0183-0

[6] S. Chilukuri and A. Sahoo, "Delay-aware TDMA scheduling for multi-hop wireless networks," in *Proceedings of the 2015 International Conference on Distributed Computing and Networking*, ser. ICDCN '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2684464.2684493

[7] P. Djukic and S. Valaee, "Delay aware link scheduling for multi-hop TDMA wireless networks," *IEEE/ACM Transactions on networking*, vol. 17, no. 3, pp. 870–883, 2008.

[8] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 270288. [Online]. Available: https://doi.org/10.1145/3341302.3342080

[9] M. Gupta, A. Rao, E. Visotsky, A. Ghosh, and J. G. Andrews, "Learning link schedules in self-backhauled millimeter wave cellular networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 12, pp. 8024–8038, 2020.

[10] S. Chinchali, P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone, and S. Katti, "Cellular network traffic scheduling with deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[11] J. Boyan and M. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," *Advances in neural information processing systems*, vol. 6, pp. 671–678, 1993.

[12] R. Glaubius, T. Tidwell, C. Gill, and W. D. Smart, "Real-time scheduling via reinforcement learning," in *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence*, ser. UAI'10. Arlington, Virginia, USA: AUAI Press, 2010, p. 201209.

[13] G. R. Ghosal, D. Ghosal, A. Sim, A. V. Thakur, and K. Wu, "A deep deterministic policy gradient based network scheduler for deadline-driven data transfers," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 253–261.

[14] 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 *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1684–1692.

[15] P. Corbaln, R. Marfievici, V. Cionca, D. O'Shea, and D. Pesch, "Into the SMOG: The stepping stone to centralized WSN control," in *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, 2016, pp. 118–126.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[17] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[18] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[19] openai, *Gym: A toolkit for developing and comparing reinforcement learning algorithms*, (last accessed December, 2020). [Online]. Available: https://gym.openai.com/

[20] I. Kostrikov, "Pytorch implementations of reinforcement learning algorithms," https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.