Deep Reinforcement Learning based Deterministic Routing and Scheduling for Mixed-Criticality Flows

Hao Yu, Tarik Taleb, *Senior Member, IEEE,* and Jiawei Zhang

***Abstract*—Deterministic networking (DetNet) has recently drawn much attention by investigating deterministic flow scheduling. Combined with artificial intelligent (AI) technologies, it can be leveraged as a promising network technology for facilitating automated network configuration in the Industrial Internet of Things (IIoT). However, the stricter requirements of the IIoT have posed significant challenges, that is, deterministic and bounded latency for time-critical applications. This paper incorporates deep reinforcement learning (DRL) in Cycle Specified Queuing and Forwarding (CSQF) and proposes a DRL-based Deterministic Flow Scheduler (Deep-DFS) to solve the Deterministic Flow Routing and Scheduling (DFRS) problem. Novel delay aware network representations, action masking and criticality aware reward function design are proposed to make Deep-DFS more scalable and efficient. Simulation experiments are conducted to evaluate the performances of Deep-DFS, and the results show that Deep-DFS can schedule more flows than the other benchmark methods (heuristic-based and AI-based methods).**

***Index Terms*—IoT, Industrial Internet of Things (IIoT), deterministic networking, deep reinforcement learning, mixed-criticality flows, 6G, and artificial intelligence.**

I. Introduction

THE Industrial Internet of Things (IIoT) adopted in manufacturing and factory automation is typically implemented by a specialized network for data exchange among sensors, actuators and other production equipment. To facilitate information exchange, industrial networks have evolved over the years and are expected to satisfy the emerging and challenging requirements of the new operation contexts [1]. On the one hand, conventional network technologies could not provide deterministic and efficient communications for the industrial needs. To support critical data flows generated by IIoT applications with bounded low latency and low jitter, the IEEE Time-Sensitive Networking (TSN) and the IETF Deterministic Networking (DetNet) work group have been initiated to study timing guarantee for critical traffic. On the other hand, with the huge amount of the ever-increasing IIoT connectivity, the network administrators need to rely on humans to design, configure and manage sophisticated and dynamic industrial scenarios, which is not efficient and sustainable. Next-generation network automation represented by artificial intelligence (AI) based technologies is proposed to tackle this challenge. Along with the advent of the network programmability of 5G networks, the AI-enabled paradigm will carry out the intelligent automated network configuration, optimization and management in the Industry 5.0 era.

Recently, the IETF DetNet working group has been studying deterministic data transmission by incorporating Segment Routing (SR) in Layer 3 to extend TSN technologies for queuing and scheduling, in order to support deterministic bounded latency and jitter for time-critical traffic [2]. In particular, regarding the strength of programmability of SR, the working group is currently specifying a Cycle Specified Queuing and Forwarding (CSQF) mechanism [3] to schedule the flows in a more flexible and scalable way, where the forwarding time slot can be specified for the packets, which will increase the bandwidth efficiency. In CSQF, multiple queues in the output port open in a round-robin way and transmission cycles repeat periodically at each port. By defining the segment routing identifiers (SIDs) in IP packets, it can determine the packet routing and forwarding at each hop, specifically, deciding the routing and transmission time slots along the selected path for all packets of time-critical flows, so that the end-to-end latency is controlled in a deterministic way. We refer to it as the Deterministic Flow Routing and Scheduling (DFRS) problem in this paper. To this end, a network controller is required to collect the overall network information for deciding the proper scheduling for flows.

Different kinds of solutions have been proposed to solve the flow scheduling problem: solver-based methods, e.g., an ILP tool [4] or heuristic-based methods, e.g., list-based methods [5]. Nevertheless, due to the fact that the high computational complexity of solver-based methods and heuristic-based methods are usually handcrafted with certain expertise, scalable and intelligent scheduling approaches are desired to solve the flow scheduling problem. Therefore, the authors in [6] leveraged deep reinforcement learning (DRL) to solve the time-triggered (TT) flow scheduling problem incrementally. However, the agent trained in [6] was only used to make routing decisions, the transmission cycles for time-critical packets were determined by a heuristic-based method, which would choose the earliest time slots along the path for packet forwarding to minimize the end-to-end (E2E) delay of the TT flow. Nevertheless, the users of time sensitive networks only care about the delay bound guarantee, any earlier delivery of any particular packet is not necessary. In addition, it will cause network congestion to always choose earliest available time slots to minimize the E2E delays without considering different

This research work is partially supported by the European Unions Horizon 2020 Research and Innovation Program through the Charity and Accordion projects under Grant No. 101016509 and 871793, respectively; the Academy of Finland 6Genesis project under Grant No. 318927 and the Academy of Finland IDEA-MILL project under Grant No. 352428.

H. Yu and T. Taleb are with the Center of Wireless Communications, The University of Oulu, Finland. Email: firstname.familyname@oulu.fi.

Jiawei Zhang is with the State Key Laboratory of Information Photonics and Optical Communications, Beijing University of Posts and Telecommunications, China. E-mail: zjw@bupt.edu.cn.

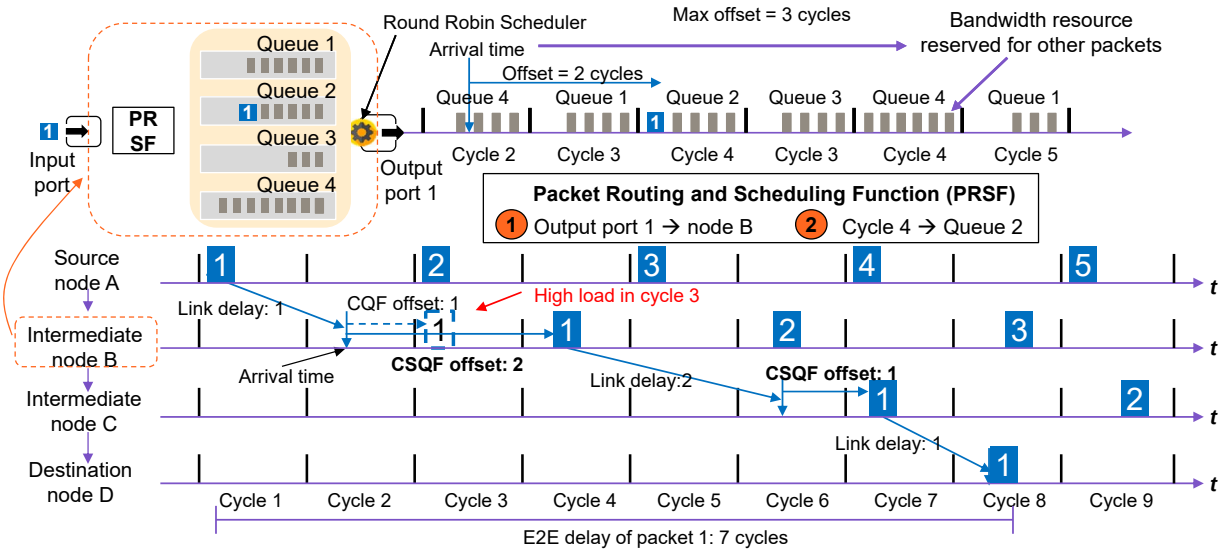


Fig. 1: CSQF-based cycle scheduling of a DN flow.

deadlines of flows.

In this paper, we will investigate on the routing and scheduling problem of mixed-criticality DetNet (DN) flows for deterministic performance guarantee and propose a deep reinforcement learning (DRL) based deterministic routing and scheduling approach to solve this problem. Note that, minimizing E2E delay is not the objective of this paper, the proposed approaches schedule the flows with the derived E2E delays near the deadlines without exhausting network resources. In addition, we will also consider the multiple criticalities of the timing requirements of DN flows, i.e., Hard Real-Time (HRT), Soft Real-Time (SRT), and train the DRL agent to make the decision on flow routing and scheduling with the objective of maximizing the number of HRT flows scheduled and the utilities of SRT flows. The contributions of this paper are shown as follows:

A DRL based-Deterministic Flow Scheduler (Deep-DFS) is devised based on a Markov decision process (MDP) approach for the flow routing and scheduling problem and a branching dueling Q-network (BDQN) is introduced into the framework to derive the optimal policy of the MDP model.

We propose several methods to enhance the efficiency and schedulability of Deep-DFS: 1) we divide a complete flow schedule into multiple simple actions to increase the scheduling scalability; 2) we use a latency aware network representation method to better extract key information for flow routing and scheduling; 3) we introduce action masking to filter invalid actions to avoid too many negative rewards; 4) we design a criticality aware reward function to schedule the flows with different criticalities. Finally, an extensive performance evaluation is carried out with both single path and multipath scheduling. The results show that, in an incremental scenario, the Deep-DFS scheme can schedule more DN flows than other benchmark methods and multipath scheduling has better performance than single path scheduling.

We introduce some related work in Section II. The system model and problem formulation are presented in Section III. Section IV illustrates the details of BDQN based deterministic flow routing and scheduling methods. The evaluation of the proposed method is discussed in Section V. Finally, we draw some concluding remarks in Section VI.

II. Related Work

In the context of TSN, most studies in the literature focus on the static scheduling and dynamic reconfiguration of gate openings and closings at output ports to satisfy a certain traffic matrix. In this case, routing information is generally given by the spanning tree protocol operating at layer 2. For 802.1Qbv, the disadvantages of the flow-based Time-Aware Shaper (TAS) are limitation of the gate control list (GCL) synthesis solution space and the long time it takes to solve the GCL synthesis in the case of large-scale networks. To solve this problem, the authors in [7] proposed a stream-based, class-based TAS without per-flow scheduling, which relaxes the assumption that gate openings for multiple ST queues are enforced to be mutually exclusive. Furthermore, the authors in [8] proposed a more general flexible window-based scheduling model, i.e., besides the above constraint relaxation, windows do not have to be aligned and can be placed at any time slot on nodes in networks. For network reconfiguration under dynamically changing requirements, the concept of multi-stream gate control for TAS was proposed by [9]. The proposed idea enabled runtime reconfiguration of the GCL to avoid reduction in bandwidth utilization irrespective of the burst size and the number of streams.

In case routing can be also decided, e.g., in layer 3, the joint routing and scheduling problem remains a challenge to be tackled. The authors in [10] present a formulation in the integer linear programming (ILP) framework which models the joint routing and scheduling problem for flows of periodic real-time transmissions in converged TSN networks. Network calculus-based flow routing and bandwidth allocation in IP-over-WDM

architecture is also investigated to achieve the deterministic data delivery in metro-aggregation networks [11]. The authors in [3] also focus on the joint routing and scheduling problem in large-scale deterministic networks using CSQF to maximize traffic acceptance for network planning and online flow admission. Joint routing and network resource allocation for the deterministic service function chaining (SFC) problem was also investigated in [12], where a novel Deterministic SFC Deployment algorithm (Det-SFCD) and an SFC Adjustment algorithm (Det-SFCA) were proposed to ensure deterministic performance during the SFC lifetime.

To the best knowledge of the authors, limited work has been done on AI-based methods for deterministic flow routing and scheduling and the problem of deterministic latency (not minimizing the E2E latency) provisioning upon the mix-criticality flows has been solved. Therefore, we use DRL to solve the deterministic flow routing and scheduling problem from this perspective, which is different from the above-mentioned work.

III. System Model and Problem Formulation

A DetNet system refers to a network comprised of DetNet nodes (i.e., routers), whereby the packet forwarding delay inside a node is deterministic and known through a centralized flow scheduling scheme. In a DetNet system, the delay induced by forwarding a packet comprises of four parts: (i) propagation delay, which is decided by the physical distance between two nodes; (ii) processing delay, which is related to the procedure of receiving the packets and sending them to the upper layers for routing and scheduling decision; (iii) transmission delay, which is the time for putting the packet on the physical link; and (iv) queuing delay, which refers to the waiting time in the queue of the output port because of the accumulation of packets from different input ports to the same output port [13]. If the topology information (i.e., distance between any nodes and bandwidth of physical link) is given, the propagation, processing and transmission delay can be assumed to be constant. Thus, to make the overall forwarding delay to be deterministic, the queuing delay should be determined properly in advance, ensuring the sum of delays along the path (end-to-end delay) equal to a constant and is within the latency requirements.

A. CSQF enabled DetNet system

Initially, Cycle Queuing and Forwarding (CQF [14], i.e, IEEE 802.1Qch) is proposed as a peristaltic shaper which considers 2 queues on ports to be open and closed alternatively in a cyclic fashion. It divides the time into different *cycles* with an equal duration T. A packet sent from the precedent node in cycle c must be received during the same cycle in the subsequent node and then transmitted in cycle $c+1$. Although CQF can control well the delay over each hop (at most two cycles), the scalability of this mechanism is not enough since it only works well for small networks and assumes perfect synchronization between nodes.

To improve scalability and flexibility, the Cycle Specified Queuing and Forwarding (CSQF) mechanism [15] has been devised as an emerging standard draft from the IETF DetNet working group as the evolution of the CQF mechanism. CSQF is proposed to delay packets with more queues and specify a certain cycle to transmit packets. Inside a CSQF-enabled router, N queues will be equipped in each output port and N_D queues out of N ($N_D \quad N$) queues are reserved for time-critical traffic, while the remaining Non-critical (NC) queues are for best effort (BE) traffic. These N queues transmit packets in a round-robin fashion, that is, during each *cycle*, only one queue is active for emitting a packet to the physical link, the other ($N \quad 1$) inactive queues are closed and enqueue packets for future transmissions. Note that the number of packets that are enqueued in each inactive queue is related with the buffer size of each queue, and improper enqueuing will incur packet loss. The N_D time-sensitive queues are dedicated to the time-critical flows by resource reservation. The assignment of packets to specific queues actually decides their transmission cycle, and a packet can be delayed by at most ($N \quad 1$) cycles. This assignment can be determined by a centralized controller in advance, while the BE flows without critical timing requirements will not be scheduled in advance by the controller. When the packets of BE flows arrive at each node, they will be directly inserted into the ($N \quad N_D$) NC queues, whose queuing delay is not controllable or deterministic. Note that unlike CQF, CSQF operates at layer 3 [15], as it allows to specify the routing and cycle scheduling of packets (e.g., with Segment Routing).

B. DetNet and Flow model

Network topology in this paper is modeled as a directed graph $G = (V, E)$, where V is the set of nodes representing DetNet enabled routers. The nodes are connected with data links represented by the directed edge set $E \quad V \quad V$. If there exists a physical link between $u, v \in V$, then $(u, v), (v, u) \in E$. Each edge $e_i = (u, v) \in E$ induces a delay d_{e_i} which comprises its propagation delay, transmission delay as well as processing delay. Time is partitioned into *cycles* of equal duration T, and T represents the set of cycles. One cycle is the minimal scheduling unit that packets can be inserted into and defines the granularity of latency calculation.

A DetNet (DN) flow is defined as a periodic unicast traffic from a source node to a destination node. We denote the set of DN flows as F to be scheduled within the network G. A DN flow $f_k \in F$ is defined as a tuple $(src_k, dst_k, prd_k, bw_k, D_k^{max}, D_k^{min})$, where

- src_k and dst_k represent the source and destination nodes of flow f_k.
- prd_k denotes the period of flow f_k, which means the source node sends the packets every prd_k cycles.
- bw_k is the total traffic that the source node emits in one cycle.
- the delay experienced by packets should be larger than a minimum delay bound D_k^{min} and smaller than a maximum delay bound D_k^{max}, as then the jitter does not exceed ($D_k^{max} \quad D_k^{min}$).

Since flows are featured with the different period prd_k, we define an overall scheduling cycle, which is referred to as hypercycle, so that all network behaviors are the same in each hypercycle. The hypercycle prd_s of all flows can be calculated

TABLE I: Notation and variables

Notation	Description
Topology	
G	Substrate networks
V, E	Set of nodes and edges in substrate networks
u, v	Physical nodes in the network G
e_i	Physical links in the network G
d_{e_i}	Link delay of edge e_i
T	Set of cycles
T, cap	Cycle duration & Cycle capacity
Service requests	
F	Set of DN flows
f_k	DN flow k
src_k, dst_k	Source and destination node of flow k
prd_k	Period of flow k
bw_k	Traffic load of flow k within one cycle
D_k^{max}, D_k^{min}	Maximal & minimal delay bound for flow k
$U_k(t)$	Utility function of a SRT flow k with t
H_k	If HRT flow k is scheduled successfully
prd_S	Least common multiple of periods of all flows
Decision Variables	
$o_{k;e_i}$	Transmission offset of flow k on edge e_i
e_j^k	Edge on which flow k is routed
t_i^k	Index of cycle of edge e_i on which the packets of flow k are transmitted
DRL-related notations	
s_t	Network state at time t
a_i^k	The ith sub-action for flow k
$R(a_i^k)$	Reward of the ith sub-action for flow k
U_{e_i}	Link utilization rate of e_i
U_S	Global link usage of topology
$Q_{e_i}^t$	Whether cycle t on e_i is fully occupied
$I_{e_i}^t$	Cycle utilization rate of t on e_i
I_{e_i}	Overall cycle utilization rate of edge e_i
$P_{e_i}^t$	Traffic load in cycle t of edge e_i
n_d	Number of sub-actions of the dth dimension
$A_d(s_t, a_i^k)$	Advantage of the dth dimension
$V(s_t)$	Common state value
M	Action dimension
System Parameters	
α, β, η	Reward coefficients
γ	Discount factor
N, N_D	The number of total and time-sensitive queues within a node

as the *least common multiple* of the periods of all flows. Hence, we will discuss the flow scheduling problem in one hypercycle. Actually, the starting time of the hypercycles at different nodes may be not synchronized and there exists an offset between two nodes due to clock drift, which can be measured and known by the controller. For the sake of simplicity and without loss of generality, we assume there is no offset so that all hypercycles are aligned across the networks.

Furthermore, considering criticalities in terms of latency requirements, the DN flows can be further classified into: hard real-time (HRT) flows which have strict deadlines, and soft real-time (SRT) flows whose QoS can be downgraded due to deadline violation. Best effort flows, which have no timing requirements, will be also considered in this paper as background flows. Both HRT and SRT flows have the delay bounds D_k^{max} and D_k^{min}. However, the HRT delay bound is *hard*, and if the delay bounds are violated, it may result in catastrophic consequences. The scheduling policy must guarantee that all HRT flows in the networks are transmitted within the delay bounds. The SRT deadline is *soft*, that is, the performance of the SRT flow will degrade if the delay bounds are missed. Similar to [16] [17], a positive utility function is introduced to evaluate the performance of SRT flows, denoted with $U_k(t)$, whereby t is the actual experienced E2E delay. If the packets of a SRT flow arrive within the delay bounds, the utility keeps on a predefined positive value (maximal value). The utility function decreases to zero with an actual E2E delay when it goes beyond the delay bounds, as specified by the definition of the utility function $U_k(t)$.

C. Deterministic Flow Routing and Scheduling (DFRS): an example

To ensure that no collision or congestion can happen, the controller needs to decide, for each packet, where and when it will be transmitted in each node, i.e., if a packet is sent in the first available cycle or delayed by one or more additional cycles before transmission.

In Fig. 1, we show an example of how a packet is propagated from node A to node D through node B and C. We assume that 1) the link delays of $d_{A;B}$, $d_{B;C}$ and $d_{C;D}$ are one cycle, two cycles and one cycle, respectively; 2) the period of the flow of interest (FOI) is 2 cycles. Once the packets of FOI are sent from A, they are received at B in the next cycle (e.g., packet 1 is sent in cycle 1 at node A and received in cycle 2 at node B), since the link delay between node A and B is one cycle. Upon receiving packet 1 in cycle 2, the controller can decide to forward packet 1 in the next cycle (cycle 3). However, considering the high traffic load in cycle 3 of Node B, it is better to choose to delay the packet forwarding by 2 cycles (i.e., CSQF offset), that is, packet 1 is forwarded in cycle 4. Then it is received at cycle 6 due to two cycles delay between node B and C. The E2E delay of a packet is calculated as the number of cycles it costs along the path. For example, the E2E of packet 1 is 7 cycles (cycle 8 - cycle 1). What the controller should accomplish is, on the one hand, to ensure that the E2E delay of packets are within the delay bounds of this flow; on the other hand, to avoid the network congestion on certain cycles or edges.

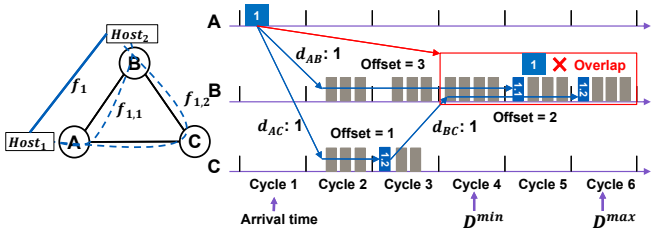


Fig. 2: CSQF-based multipath scheduling.

D. DFRS: Multipath Case

For more efficient flow scheduling and higher load balancing, Multipath TCP (MPTCP) technology [18], which allows multiple paths in a single TCP connection by spreading traffic data across multiple parallel sub-flows, has shown great advantages in emerging scenarios where heavy traffic needs to be transmitted. Different from the single-path flow scheduling in Fig. 1, flow splitting and multipath routing are considered in this scenario, where one single communication path is not

sufficient to transmit the DN flow and multi communication paths become needed. As shown in Fig. 2, we assume that a TCP connection f_1 is upcoming between Host 1 and Host 2, whose maximum and minimum deadline are 5 and 3 cycles. If the scheduler fails to find a valid schedule along the single path, for example, there is not enough bandwidth for flow 1 within cycles 4-6 in node B, multipath scheduling will be applied to this flow by splitting it into multiple sub-flows evenly. For instance, there are two candidate routing paths $f(A;B)g$ and $f(A;C);(C;B)g$ which can be leveraged for two sub-flows, i.e., $f_{1:1}$, $f_{1:2}$. Eventually, the end-to-end delay of these two sub-flows are 4 and 5 cycles separately, and packets of sub-flows will be reassembled at Host 2 without violating the deterministic delay requirement of flow f_1.

E. DFRS modelling

For a flow f_k to be scheduled, the controller needs to determine a unique feasible scheduled path P, i.e. a sequence of edges (e_1,e_2,...,e_n) where edge e_0 starts from src_k and edge e_n ends at dst_k. e_i and e_{i+1} are adjacent edges. We should have the constraints:

$$e_0{:}src = src_k{:} \tag{1}$$

$$e_n{:}dst = dst_k{:} \tag{2}$$

$$e_i{:}dst = e_{i+1}{:}src{:} \tag{3}$$

However, at each edge e_i, it is impossible to determine the transmission cycles for every packet of this target flow due to the high scheduling complexity. We define an integer variable $o_{k;e_i}$ to represent the offset at edge e_i for all packets of flow f_k, that is, all packets of flow f_k arrived at edge e_i will be delayed by $o_{k;e_i}$ cycles. If we assume that the first packet of flow f_k leaves the source node src_k at t_1^k, then it will arrive at the next node on the cycle $t_1 + d_{e_1}$, and will be transmitted again on cycle $t_1^k + d_{e_1} + o_{k;e_i}$, denoted by t_2^k. Therefore, the cycle determination for flow k can be represented by an integer sequence $(t_1^k;t_2^k;:::;t_n^k)$, where $t_i^k \ 2 \ Z^+$ indicates the index of cycle that the first packet is supposed to be forwarded at the corresponding node e_i. The transmission cycles of the remaining packets can be calculated by $t_i^k + l \quad prd_k$, $l \ 2 \ f0;1;:::;ng$ easily.

Thus, the schedule S_k of a DN flow f_k from source node src_k to destination node dst_k can be denoted as $f(e_1^k;t_1^k);(e_2^k;t_2^k); \quad ;(e_n^k;t_n^k)g$. An S_k is valid for flow f_k if the following two conditions hold.

(1) E2E latency constraint:

The E2E delay of all packets of HRT flow f_k must not exceed the maximum and minimum end-to-end delay bounds D_k^{max} and D_k^{min}.

$$\textbf{C1: } D_k^{min} \quad t_n^k \quad t_1^k \quad D_k^{max}{:} \tag{4}$$

(2) Cycle capacity constraint:

If the packets of flow f_k are decided to be transmitted at edge e_i at cycle t, denoted by $x_{k;e_i}^t$, the bandwidth capacity bw_k in the corresponding cycles will be reserved. Since the bandwidth capacities of cycles are shared among the scheduled flows, the traffic load at any edge e_i during any cycle t must not exceed its capacity cap, which is the value of cycle duration T multiplied by link data rate G. This condition is ensured by the constraint:

$$\textbf{C2: } \sum_{f_k \in D} x_{k;e_i}^t \quad bw_k \quad cap; \ 8e_i \ 2 \ E;8t \ 2 \ T: \tag{5}$$

F. Problem formulation

The flow scheduling problem can be formulated as: given the network topology and DN flows, finding valid schedules (route and cycle allocation) for all flows so that all HRT flows are scheduled and the utility of SRT flows are maximized. We define the variable H_k to indicate if the HRT flow f_k is successfully scheduled: $H_k = 1$ when HRT flow k is successfully scheduled, 0 otherwise. The Deterministic Flow Routing and Scheduling (DFRS) problem can be defined as follows:

$$\max \sum_{f_k \in F} \begin{cases} H_k & \text{if } f_k \text{ is HRT flow} \\ U_k & \text{if } f_k \text{ is SRT flow} \end{cases} \tag{6}$$

$$s{:}t{:}\ \textbf{C1,C2}{:}$$

G. Markov Decision Process Based Model

The learning process of Reinforcement learning (RL) is usually modeled as a Markov Decision Process (MDP), with the state space S, the action space A, and the reward R devised as follows.

1) State space: A system state s_t represents all the information of the whole network that the RL agent can observe and use to generate a schedule S_k for a flow f_k. We denote network state s_t by extracting the network features from three aspects: 1) topology information, 2) flow information, and 3) network load information. s_t can be divided into $s_t = fs_{t;e_1};s_{t;e_2};:::;s_{t;e_i};e_i \ 2 \ Eg$ and each $s_{t;e_i}$ consists of:

- If the edge e_i is adjacent to the edge of the former action or to the source node src_k.
- The distance between this edge and destination node dst_k.
- The difference between the current selected cycle and d_k^{min}, (i.e., minimum delay requirement-passed delay).
- The difference between the current selected cycle and d_k^{max}.
- The traffic load of this edge, which is denoted as the ratio of the number of available cycles to the total number of cycles. Generally, choosing an edge with lower network load can maintain load balancing and avoid bottleneck links.
- The list of cycle load (in percentage). The state s_t will be updated each time after the agent selects a sub-action.

2) Action space: Deep-DFS improves the scalability by dividing the schedule of a flow into a set of sub-actions. That is, the schedule $S_k = fa_1^k;a_2^k;:::;a_n^kg = f(e_1^k;t_1^k);(e_2^k;t_2^k);:::;(e_n^k;t_n^k)g$, where $a_i^k = (e_i^k;t_i^k)$, of flow k is derived from a sequence of edges and cycles. Specifically, each $a_i^k = (e_i^k;t_i^k)$ acts as a sub-action, e_i^k denotes the edge that a flow needs to go through, and t_i^k represents the cycle that packets are transmitted on e_i^k. A valid path should satisfy the following conditions: 1) the edges in sub-actions should connect in head-to-tail; 2) the constraint $t_{i+1}^k > t_i^k + d_{e_i}$ should

be kept, where d_{e_i} is the link delay of e_i, to ensure a valid timing. Combined with constraint (4) and (5), a valid schedule for flow f_k should meet these four constraints.

3) Reward Function: After receiving a sub-action a_i^k, the agent will obtain a reward value $R(a_i^k)$ from the environment based on the effect that this sub-action causes.

The reward of a sub-action a_i^k will comprise of two parts: 1) how much congestion it brings to the network; 2) whether this sub-action will finalize a complete schedule of a flow, which is also valid.

We define the link utilization rate U_{e_i} as the number of cycles which are not available for flows divided by the number of all cycles on link e_i

$$U_{e_i} = \frac{\sum_{t \in T} Q_{e_i}^t}{|T|}. \tag{7}$$

where $Q_{e_i}^t$ represents whether cycle t on e_i is fully occupied, $|T|$ and $|E|$ are the total number of cycles considered in one hypercycle and the number of edges in the topology. Furthermore, the global bandwidth utilization ratio U_s over all cycles and edges in the network is defined as

$$U_s = \frac{\sum_{e_i \in E} \sum_{t \in T} Q_{e_i}^t}{|T||E|}. \tag{8}$$

Besides evaluating the impact of a selected sub-action on the link utilization rate, the cycle utilization rate $I_{e_i}^t$ should be also considered, which is defined as

$$I_{e_i}^t = \frac{\sum_{l=0}^{\frac{prd_s}{prd_k}-1} P_{e_i}^{t+l\,prd_k}}{\frac{prd_s}{prd_k}\,cap}. \tag{9}$$

where $P_{e_i}^t$ denotes traffic load in cycle t of edge e_i. Thus, the overall cycle utilization rate of edge e_i can be defined as

$$I_{e_i} = \frac{\sum_{t \in T} P_{e_i}^t}{|T|cap}. \tag{10}$$

In order to make the training converge fast, we use $\ ;\ ;\ $ to adjust the weight of each part making the reward value within $(\ \ 1; 1)$, and then the reward of the sub-action a_i is denoted as

$$R(a_i^k) = \ \ (U_s \ \ \ U_{e_i}) + \ \ (I_{e_i}^t \ \ \ I_{e_i}). \tag{11}$$

Note that if the usage U_{e_i} is larger than the global usage U_s after mapping the flow to edge e_i, it means a negative reward for this sub-action a_i. The same applies for cycle usage I_{e_i} and $I_{e_i}^t$.

The second part takes effect only if the sub-action a_i^k is the last edge of a valid path for a flow. If the agent finalizes the scheduling of a flow, H_k or U_k will be calculated according to the flow types.

After selecting the last action a_n^k of a valid route, we will check if this flow is scheduled successfully (i.e., if the latency, capacity and routing requirement are all satisfied), and then the rewards for the sub-actions $R(a_1^k); R(a_2^k); \ldots; R(a_n^k)$ will be updated by adding an extra reward for the second part in a decayed way. Intuitively, earlier sub-actions have a larger exploration space, and thus pose less impact on a valid schedule, while the latter sub-actions are more significant for constituting a valid schedule.

$$\hat{R}(a_i^k) = R(a_i^k) + \ \ f H_k; U_k g \ \ \ \frac{i}{n}; 8i \ 2 \ f1; \ldots; ng. \tag{12}$$

H. Optimization formulation

This paper aims to obtain the optimal deterministic flow routing and scheduling policy, denoted by , to maximize the long-term rewards of mapping flows into networks. The DFRS problem can be transferred to the optimization problem which maximizes the expected future discounted rewards as follows:

$$max\ \mathbb{E}\Big[\sum_{f_k \in F}\sum_{i=0}^{\infty} {}^{i} R(a_i^k)\Big]. \tag{13}$$

where $R(a_i^k)$ is the reshaped reward under the policy from (12), and the discount factor indicates how much the current rewards are valuable than later rewards.

To solve the optimization problem in Eq. (13), we propose a branching dueling q-network based deterministic flow routing and scheduling algorithm which will evaluate each action dimension (i.e., edge and cycle selection) separately, and also use action masking to improve the training efficiency.

IV. Branching Dueling Q-Network Based Deterministic Flow Routing and Scheduling

To facilitate the learning process for a MDP problem, deep Q-Network (DQN)-based methods are widely leveraged in the literature. For example, Dueling DQN is proposed to eliminate overestimation in the learning process and improve the performance of the double deep Q-network (Double DQN) algorithm [19] [20]. By applying a primary neural network Q_{net} as a nonlinear function approximator to select an action, and using a target neural network Q_{target} to estimate the target Q-value of the taken action, Double DQN stabilizes the training process of the RL agent. Dueling DQN further improves Double DQN by using two separate neural networks to estimate the state value and the action advantage, and then the state values and the action advantages are aggregated at the output layer. By doing this, Dueling DQN can perform more robust estimations on state values, which lead to significant improvements on convergence rate and the stability of the learning process.

However, in order to solve the DFRS problem defined in this paper using the Dueling DQN method, several challenges still remain to be tackled:

Unlike other simple scheduling tasks where the action space is featured with only single dimension, the action space of DFRS problem is with two dimensions, i.e., edge and cycle selection, which are independent from each other. However, these two dimensions are synergistic when scheduling DN flows for deterministic performance. The edge selection should consider the cycle utilization of network and cycle selection also depends on the selected edge/path, which brings a huge challenge to deterministic flow scheduling;

To avoid a large amount of invalid actions for scheduling a DN flow and improve learning efficiency, the size of

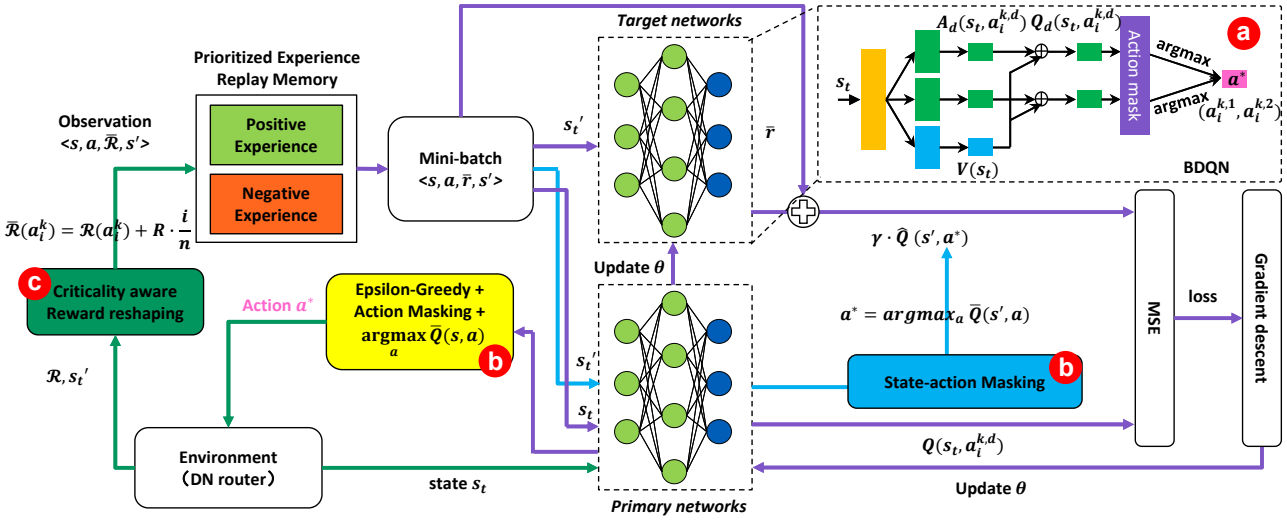


Fig. 3: Branching dueling Q-networks based learning process.

the action space, which is proportional with the amount of the edges in the topology and cycles considered in a hypercycle, can be further reduced;

Specialized rewards for flows with different criticalities should be designed so that the DN flows are scheduled with different priorities.

Therefore, as shown in Fig. 3, three approaches or techniques are proposed in this section to respond to the challenges mentioned above. Generally, two neural networks are employed, the *Primary network* for selecting an action, and the *Target network* for estimating the target Q-value of the taken action. The parameters of target networks will be updated from primary networks every certain number of iterations. The experience replay technique is also adopted to stabilize the learning process. In addition, to improve the learning efficiency and accuracy, we carry out the following:

We incorporate the advances of action branching with dueling deep Q-network (Dueling DQN) [21], which is referred to as a branching dueling Q-network (BDQN), to solve the action selection with multiple dimensions;

We use the action masking to block a large amount of invalid actions. A delay-aware masking method is designed to reduce the size of the action space while ensuring enough exploration space;

We design a criticality-aware reward function to entitle different priorities for DN flows with different types, e.g., high priority for HRT flows.

A. Branching Dueling Q-Network

The action branching methods proposed in [22] improve the conventional deep Q-network to solve MDP with multi-dimensional discrete action space. The main idea is to evaluate the individual action dimension $a_i^{k;d}$ 2 A_d; d 2 $f1;:::;Mg$, e.g., edge and cycle selection in the DFRS problem, while keeping a common state-value estimator between multiple dimensions. Each dimension has a certain degree of autonomy. The structure of BDQN is illustrated in Fig. 3(a), the BDQN further splits the advantage branch into two advantage branches based on Dueling DQN, while keeping a shared representation of the input state. Specifically, the advantage branches correspond to the two dimensions of action in this study, i.e., $a_i^{k;1} = e_j^k; a_i^{k;2} = t_i^k$, each dimension has n_d sub-actions. For example, the n_d of the edge dimension is jEj. As shown in Fig. 3(a), a network state s_t is input into a shared neural network (yellow block), which will then compute a latent representation which is used for the evaluation of the state value (blue block) $V(s_t)$ and the factorized (state-dependent) action advantages (green block) $A_d(s_t; a_i^{k;d})$ on the subsequent independent branches. The dimensions of action advantages and state value are aligned, i.e., $maxfn_1;:::;n_Mg$. Then state value $V(s_t)$ and the factorized advantages are combined to output the Q-values for each action dimension. Note that, in order to filter the Q-values for the valid sub-action of each action dimension, *action masking* is applied here to accelerate the learning processing. Finally, the sub-actions with maximal Q-values of each action dimension are selected for the generation of a joint action tuple.

The advantage of each dimension, i.e., $A_d(s_t; a_i^{k;d})$ is trained with the common state value $V(s_t)$, and then the Q-value of each dimension $Q_d(s_t; a_i^{k;d})$ is obtained by aggregating the value branch and the corresponding advantage branch as follows.

$$Q_d(s_t; a_i^{k;d}) = V(s_t) + (A_d(s_t; a_i^{k;d}) - \frac{1}{n_d} \sum_{\hat{a}_i^{k;d} 2 A_d} A_d(s_t; \hat{a}_i^{k;d})): \tag{14}$$

Then, the action selection follows the -greedy policy, that is, select a random action with probability and with probability (1) to select:

$$a = f\arg\max_{a_i^{k;1}} Q_1(s_t; a_i^{k;1};); :::; \arg\max_{a_i^{k;M}} Q_d(s_t; a_i^{k;M};)g \tag{15}$$

The TD-target of BDQN is similar with that in dueling DQN to avoid maximization bias, but it is derived by averaging

across all dimensions of the action

$$y = \hat{R}(a_i^k) + \frac{1}{M} \sum_d \hat{Q}_d(s_t^0, \arg\max_{\hat{a}_i^{k,d} \in A_d} Q0_d(s_t^0, \hat{a}_i^{k,d})) \quad (16)$$

We use Mean Square Error (MSE) to update the parameter by the gradient descent method.

B. Action Masking

At the early stage of training, the agent learns to find a valid schedule of a flow in an exploring way. The sub-action is selected by the agent for the action space $|E| \quad |T|$, most edges and cycles in the action selected at this stage are not valid for constituting a feasible path, which will slow down the learning efficiency. To avoid the sparse rewards induced by the invalid actions in the training processing, action masking is proposed in this section to block the actions which are obviously invalid to improve the learning efficiency.

Action masking will be applied in two phases of the whole training processing: 1) the action selection phase and 2) the Q value updating phase. We maintain binary lists $[a_i]; [c_j]$, where $i \in E$ and $j \in T$ to filter the invalid actions each time the agent selects a sub-action. $a_i = 1$ if i^{th} edge is adjacent to the edge selected by the former action, otherwise $a_i = 0$. When selecting valid cycles along the path, two aspects should be considered: 1) maximum delay (offset) in each switch and 2) residual available latency budget. Since the queues in a router transmit the packets in a round-robin way, which means each queue transmits packets every N cycles, the packets will be delayed at most $(N \quad 1)$ cycles in a switch, that is $c_j = 1$ for $j \in (t; t + N \quad 1)$ should be satisfied, where t is the cycle selected for the former sub-action.

Besides the constraint of maximum offset in one node, the cycle selection should also satisfy the end-to-end latency requirement. Therefore, each time the agent selects an action, the actual latency that this flow consumes should not exceed the maximum latency bound D_k^{max}. That is, $c_j = 1$ for $j \in (t; D_k^{max})$, otherwise, $c_j = 0$. The list $[a_i]; [c_j]$ is recalculated and multiplied by original Q values each time when making a sub-action decision. This narrows down the scope of action selection, the agent chooses an edge and cycle from valid actions that have the highest Q value, which can produce more positive experience and improve the sample efficiency in the training process.

Action masking is also applied in the process of Q-value updating. During the learning period, to avoid the overestimation of the actual Q-value, action selection is based on the Q value of policy network $Q(s; a)$ in the TD-error calculation as (16). In this section, we modify the $Q_d(s^0; \hat{a}_i^{k,d})$ by multiplying it with $[a_i]; [c_j]$, that is, $Q_d^0(s^0; \hat{a}_i^{k,d})$.

C. Criticality-aware reward function

Although the controller can schedule the real-time flows by assigning the cycles for packet transmission along the path with the CSQF mechanism, it cannot always guarantee all real-time flows are scheduled successfully because of the resource competition. Therefore, Deep-DFS should learn the criticalities of different DN flows, so that all HRT flows are scheduled successfully and the total utility for SRT flows is maximized. Besides H_k, which can represent if a HRT flow is scheduled, U_k can be defined by a linear function, starting at a maximum utility of 1, linearly decreasing after breaking the delay bounds, reaching a zero utility at a certain value, e.g., $U_k = 1$, when $8 < t < 10$, $U_k = 0$ when $t < 6$ or $t > 12$. Therefore, the soft delay bound of a SRT flow is denoted as $(6; 8; 10; 12)$.

TABLE II: Flow types with different criticalities

Flow type	Size	Period	Delay Bounds
F^{HRT}	$\{1,2,4\}$ DUs	$\{2,4,8,16\}$ cycles	$\{(8,10),(18,20)\}$ cycles
F^{SRT}	$\{1,2,4\}$ DUs	$\{2,4,8,16\}$ cycles	$\{(6,8,10,12), (16,18,20,22)\}$ cycles
F^{BE}	$\{2\}$ DUs	$\{2\}$ cycles	—-

V. Evaluation

A. Evaluation Settings

By conducting simulation experiments of Deep-DFS, we demonstrate the effectiveness of Deep-DFS in maximizing the number of scheduled flows under different network scales as well as flow distributions by comparing it with two benchmark methods.

We evaluate the performance of Deep-DFS on a Ladder Network Topology introduced in an Ethernet Consist Network [23], which is an international standard of train communication network. In this paper, the size of the ladder topology is varied from 6 nodes to 10 nodes. Besides the network topology, the DN flows for training and evaluation are both generated as per Table II with the probabilities of 40%, 40% and 20% on the HRT, SRT and BE types, respectively. Note that the flows of F^{BE} are considered as background traffic which has no deadlines. Within each flow, src_k and dst_k are randomly selected from all nodes in the networks. The packet length (in data units, DUs), period (in cycles) and delay bounds (in cycles) are selected randomly from the set in Table II. In addition, we also set the data rate of all physical links to 1 Gbit/s, and the capacity of each cycle is 100 DUs. The hypercycle is 16 according to periods of all flows.

For the configuration of the BDQN, the parameters are employed based on the common settings for designing neural networks [19], i.e., two-fully connected hidden layers are together deployed with input and output layers. The size of the hidden layers is 128, the size of the output layer is 2, which represents the index of the selected edge and cycle. The mini-batch size is 32 and the discount factor is set to 0.5. The initial value of is 1.0 and decays to its final value 0.01. The agent will be trained and evaluated on the same topology but with different flows, separately.

B. Baseline Methods Compared

To evaluate the effectiveness of Deep-DFS, we compare it with the following baseline methods. **1) DRLS:** The DRL-based TTEthernet Scheduler [6] outputs edge action step by step to constitute a complete flow schedule, but use a heuristic method to select the earliest available cycles with low degree. **2) HLS:** heuristic list scheduler (HLS) selects [5] a time slot

which leads to the minimum end-to-end delay on the shortest routing path between the source node and the destination node of a flow. If this minimum delay exceeds the deadline of f_k, the scheduler fails to schedule this flow.

C. Incremental Scheduling Scenario

In this scenario, we randomly generate the flow one by one and insert the flow to the network incrementally. If the agent fails to schedule a HRT flow then it stops, and then we compare the maximum number of successfully scheduled HRT flows and the utility of the SRT flows of each solution.

As shown in Fig. 4(a), Deep-DFS can schedule more HRT flows than the other two methods in general, specifically, 14.8% more on average than DRLS, 32.1% more on average than HLS in ladder networks. Since HLS always selects the first available time slot to transmit the packets on the shortest path, it will derive the minimum latency for all flows regardless of the flow criticalities and their delay bounds. Therefore, HLS will saturate the cycles and edges soon, and increase the probability of flow blocking by some fully occupied cycles. DRLS takes into account the cycle usage on the edge, it avoids selecting the cycles with high degree in order to save more bandwidth for the flows with different periods. However, DRLS still tries to select the first available cycle for packet forwarding and minimizes the E2E delay of the flow, which is in conflict with the long-term objective of maximizing the number of flows scheduled in this system. To this end, Deep-DFS redesigns the network state representation and reward function to make delay-aware decisions on edge and cycle selection, so that the HRT flows are prioritized and no bandwidth resources are wasted on minimizing the E2E delay. When the size of the ladder topology becomes larger, Deep-DFS schedules more flows (39.1% more than HLS on average) in the ladder topology with 10 nodes than with 6 nodes (29.3% more than HLS on average). This is because Deep-DFS has more exploration space in a larger topology, while HLS only selects the shortest paths to route the flows regardless of topology size.

We also evaluate the average utility of SHR flows with the percentage of BE traffic. We set the probability of generating BE traffic from 0.2 to 0.36, with HRT and SRT flows are generated with the same probability. As we can see in Fig. 4(b), it is obvious that the average utility will decrease with more BE traffic inserted in the network. With the increasing BE traffic, the network bottleneck will come earlier and the utility of SRH flows in the HLS case will decrease a little faster than the other two methods due to the selection of shortest paths, as discussed above. The link and cycle usages are also shown in Fig. 4(c) and Fig. 5(a). The results show that although the HLS method will lead to more cycles with high traffic load (60%), the link usage induced by HLS is lower than the that of Deep-DFS, which is not intuitional. This is because, on the one hand, the resources are exhausted in earlier cycles by minimizing the E2E delay with HLS, though, HLS also stops to schedule flows earlier than Deep-DFS. That makes the overall link usage of HSL lower than Deep-DFS by 21.3% on average in the ladder topology. We also find that the link usage decreases slightly with more nodes in the ladder topology for Deep-DFS, as it prefers to choose a longer route to balance the link load and avoid the bottleneck.

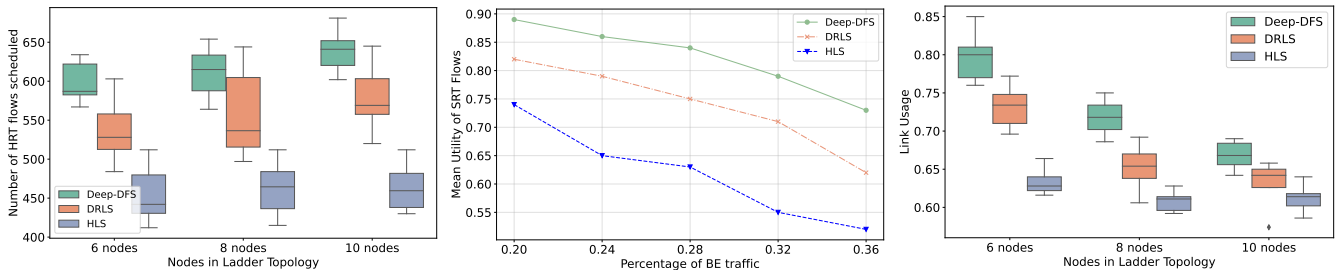
D. Multipath Scheduling Scenario

As shown in Fig. 6, to implement the multipath scheduling with a DRL solution, flow splitting module is needed in the DRL agent. The flow splitting and tagging function and the flow recovery function should be also deployed to map the multipath selection. Upon receiving the request information of flow f_k from the host, the DRL agent calculates the action for this flow. If the agent fails to find a valid schedule for flow f_k, the flow split module modifies the request information by dividing the size of the flow into $1/q$, e.g., 1/2 while keeping other requirements unchanged. Then the q sub-flows are fed into the action selection module again. If they are scheduled successfully, in other words, the q sub-flows are scheduled with different paths while the delay requirements of all sub-flows are satisfied, the corresponding flow splitting information will be tagged on the packets of this flow, and they will be reassembled in the destination host. Once the agent fails to schedule one of the q sub-flows, it stops.

We set the topology with 10 nodes and generate the flows with an average packet size (in DUs) from 1.5 to 3.5. We also assume that $q = 2$ and 1 DU is the minimal transmission unit that cannot be split any further in this case. From Fig. 5(b), we can observe that the number of scheduled HRT flows decreases with the increase of average packet size, as high-size packets consume more bandwidth resources. However, compared with a single path schedule, multipath scheduling has better performance in finding valid schedules for HRT flows. In addition, the number of scheduled flows in multipath scheduling decreases more slowly than that of single path scheduling for the reason that the HRT flows have more chance to be scheduled after they are split into multiple sub-flows. We also evaluate the performance of multipath scheduling in terms of the jitter of the SRT flows, as shown in Fig. 5(c). As there is no hard deadline for the SRT flows, the jitter of the SRT flows is usually higher than that of the HRT flows. Because once a HRT flow f_k is scheduled, the delay jitter of this flow ought to be within $(D_k^{max} \quad D_k^{min})$. We find that multipath scheduling also outperforms single path scheduling in terms of jitter performance. Furthermore, the jitters of the SRT flows with multipath scheduling can be well controlled within three cycles on average, while the jitters with single path scheduling are much larger, due to the more severe competition for resources in case of only one transmission path.

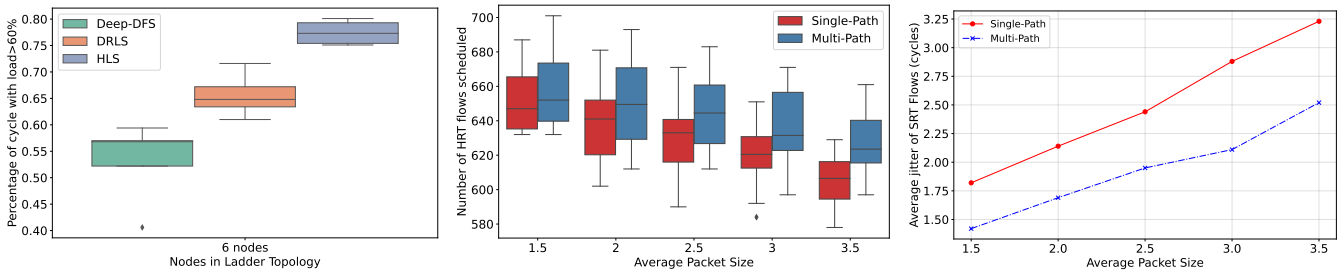
VI. Conclusion

In this study, we proposed a deep reinforcement learning based deterministic flow scheduler (Deep-DFS) to solve the scheduling problem of DetNet (DN) flows with multiple criticalities. We leverage Deep-DFS to determine the routing and cycle selection for DN flows with the Cycle Specified Queuing and Forwarding (CSQF) mechanism, where 1) the timeline is divided into multiple cycles with equal duration and 2) the controller can specify the cycles for packet forwarding so that the end-to-end delay of DN flows can be controlled effectively.



(a) (b) (c)

Fig. 4: (a) Number of HRT flows scheduled; (b) Utility of SRT flows; (c) Link Usage with different nodes in ladder topology.



(a) (b) (c)

Fig. 5: (a) Percentage of cycles with traffic load over 60%; (b) Number of HRT flows scheduled under single and multi-path scheduling; (c) Average jitter of SRT flows under single and multi-path scheduling.

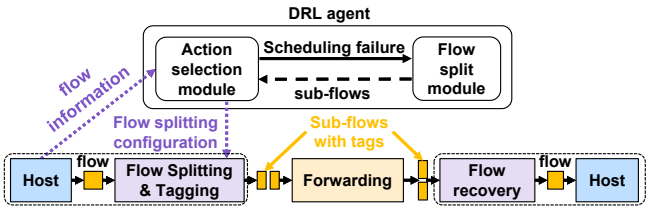


Fig. 6: DRL-based flow splitting and multipath scheduling.

To make the proposed Deep-DFS schedule the flows with multiple criticalities, several technologies were proposed to increase scalability and performance. Compared with the other AI-based and heuristic-based methods, Deep-DFS can increase the scheduled flows by 14.8% and 32.1%, respectively. However, it is worth noting that the proposed centralized scheduling approach is merely suitable for scenarios with small network scales, e.g., within a factory network. If the network scale is large, the round-trip time of the signaling between the source node of a flow and the controller becomes also too large, which makes no sense to a time-sensitive flow. Furthermore, runtime re-configuration is also a challenge for a centralized controller. Large-scale deterministic flow scheduling requires a disparate solution, e.g., distributed learning and distributed scheduling approaches, which can facilitate the learning process and network configuration locally. Therefore, the way to design a distributed learning architecture and train distributed learning agents for flow scheduling with segment routing efficiently will be considered in our future work.

REFERENCES

[1] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0," *IEEE industrial electronics magazine*, vol. 11, no. 1, pp. 17–27, 2017.

[2] H. Yu, T. Taleb, and J. Zhang, "Deterministic service function chaining over beyond 5g edge fabric," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[3] J. Krolikowski, S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang, and X. Geng, "Joint routing and scheduling for large-scale deterministic ip networks," *Computer Communications*, vol. 165, pp. 33–42, 2021.

[4] N. Wang, Q. Yu, H. Wan, X. Song, and X. Zhao, "Adaptive scheduling for multicluster time-triggered train communication networks," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1120–1130, 2018.

[5] M. Pahlevan, N. Tabassam, and R. Obermaisser, "Heuristic list scheduler for time triggered traffic in time sensitive networks," *ACM Sigbed Review*, vol. 16, no. 1, pp. 15–20, 2019.

[6] C. Zhong, H. Jia, H. Wan, and X. Zhao, "Drls: A deep reinforcement learning based scheduler for time-triggered ethernet," in *2021 International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2021, pp. 1–11.

[7] D. Hellmanns, J. Falk, A. Glavackij, R. Hummen, S. Kehrer, and F. Dürr, "On the performance of stream-based, class-based time-aware shaping and frame preemption in tsn," in *2020 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2020, pp. 298–303.

[8] M. Barzegaran, N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Real-time guarantees for critical traffic in ieee 802.1 qbv tsn networks with unscheduled and unsynchronized end-systems," *arXiv preprint arXiv:2105.01641*, 2021.

[9] Y. Nakayama and D. Hisano, "Multi-stream gate control of time aware shaper for high link utilization," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*. IEEE, 2021, pp. 1–6.

[10] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for tsn with ilp," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 136–146.

[11] H. Yu, T. Taleb, J. Zhang, and H. Wang, "Deterministic latency bounded network slice deployment in ip-over-wdm based metro-aggregation

networks," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 2, pp. 596–607, 2022.
[12] H. Yu, T. Taleb, and J. Zhang, "Deterministic latency/jitter-aware service function chaining over beyond 5g edge fabric," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2022.
[13] A. N. Abbou, T. Taleb, and J. Song, "Towards sdn-based deterministic networking: Deterministic e2e delay case," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.
[14] "Ieee standard for local and metropolitan area networks–bridges and bridged networks–amendment 29: Cyclic queuing and forwarding," *IEEE 802.1Qch-2017*, pp. 1–30, 2017.
[15] M. Chen, X. Geng, and Z. Li, "Segment routing (sr) based bounded latency," *Internet Engineering Task Force, Internet-Draft draft-chendetnet-sr-based-bounded-latency-00*, 2018.
[16] V. Gavriluţ and P. Pop, "Traffic class assignment for mixed-criticality frames in ttethernet," *ACM Sigbed Review*, vol. 13, no. 4, pp. 31–36, 2016.
[17] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft Real-Time Systems*. Springer, 2005.
[18] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," Tech. Rep., 2013.
[19] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[20] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.
[21] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.
[22] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
[23] I. E. Commission *et al.*, "Electronic railway equipment—train communication network (tcn)—part 3-4 (s): Ethernet consist network (ecn)," *International Electrotechnical Commission*, pp. 61 375–3, 2014.

Hao Yu received the B.S. and Ph.D degree in communication engineering from the Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2015 and 2020. He was also a Joint-Supervised Ph.D. Student with the Politecnico di Milano, Milano, Italy. He is currently a Postdoctoral Researcher with the Center of Wireless Communications, Oulu University, Oulu, Finland. His research interests include intelligent edge network, time sensitive networks, 6G deterministic networking.

Tarik Taleb is currently a Professor at the Center of Wireless Communications, The University of Oulu, Finland. He is the founder and director of the MOSA!C Lab (www.mosaic-lab.org). Between Oct. 2014 and Dec. 2021, he was a Professor at the School of Electrical Engineering, Aalto University, Finland. Prior to that, he was working as Senior Researcher and 3GPP Standards Expert at NEC Europe Ltd, Heidelberg, Germany. Before joining NEC and till Mar. 2009, he worked as assistant professor at the Graduate School of Information Sciences, Tohoku University, Japan, in a lab fully funded by KDDI. From Oct. 2005 till Mar. 2006, he worked as research fellow at the Intelligent Cosmos Research Institute, Sendai, Japan. He received his B. E. degree in Information Engineering with distinction, M.Sc. and Ph.D. degrees in Information Sciences from Tohoku Univ., in 2001, 2003, and 2005, respectively.

Prof. Taleb's research interests lie in the field of telco cloud, network softwarization & network slicing, AI-based software defined security, immersive communications, mobile multimedia streaming, and next generation mobile networking. Prof. Taleb has been also directly engaged in the development and standardization of the Evolved Packet System as a member of 3GPP's System Architecture working group 2. Prof. Taleb served on the IEEE Communications Society Standardization Program Development Board.

Prof. Taleb served as the general chair of the 2019 edition of the IEEE Wireless Communications and Networking Conference (WCNC'19) held in Marrakech, Morocco. He was the guest editor in chief of the IEEE JSAC Series on Network Softwarization & Enablers. He served on the editorial board of the IEEE Transactions on Wireless Communications, IEEE Wireless Communications Magazine, IEEE Journal on Internet of Things, IEEE Transactions on Vehicular Technology, IEEE Communications Surveys & Tutorials, and a number of Wiley journals. Till Dec. 2016, he served as chair of the Wireless Communications Technical Committee, the largest in IEEE ComSoC.

Prof. Taleb is the recipient of the 2021 IEEE ComSoc Wireless Communications Technical Committee Recognition Award (Dec. 2021), the 2017 IEEE ComSoc Communications Software Technical Achievement Award (Dec. 2017) for his outstanding contributions to network softwarization. He is also the (co-) recipient of the 2017 IEEE Communications Society Fred W. Ellersick Prize (May 2017), and many other awards from Japan. Some of Prof. Taleb's research work have been also awarded best paper awards at prestigious IEEE-flagged conferences.

Jiawei Zhang received the Ph.D. degree from the State Key Laboratory of Information Photonics and Optical Communications, Beijing University of Posts and Telecommunications (BUPT), China. He currently is an associate Professor with BUPT. Dr. Zhang has authored and co-authored more than 30 OFC/ECOC papers and top journal papers in optical communication and networks. His research interests include the collaboration of optical networks with IP, wireless and cloud/edge, currently with an emphasis on the advanced technologies for providing deterministic connections for future network applications. He served on the Technical Program Committees for the IEEE DRCN 2018-2020, IEEE ICNC 2017-2018, ACP2020, and for the Workshop on Cloud Computing Systems, Networks and Applications at the IEEE GLOBECOM 2014-2016, ICC 2015-2016, and INFOCOM 2017-2018 conferences. He also severed as a Guest Editor of the special issue on Resilience in future 5G Photonic Networks of Photonic Network Communications journal (Springer).