Multi-User Delay-Constrained Scheduling With Deep Recurrent Reinforcement Learning

Pihe Hu, Yu Chen, Ling Pan, Zhixuan Fang, *Member, IEEE*, Fu Xiao, *Member, IEEE*, and Longbo Huang, *Senior Member, IEEE*

***Abstract*— Multi-user delay-constrained scheduling is a crucial challenge in various real-world applications, such as wireless communication, live streaming, and cloud computing. The scheduler must make real-time decisions to guarantee both delay and resource constraints simultaneously, without prior information on system dynamics that can be time-varying and challenging to estimate. Additionally, many practical scenarios suffer from partial observability issues due to sensing noise or hidden correlation. To address these challenges, we propose a deep reinforcement learning (DRL) algorithm called Recurrent Softmax Delayed Deep Double Deterministic Policy Gradient (`RSD4`) (https://github.com/hupihe/RSD4), which is a data-driven method based on a Partially Observed Markov Decision Process (POMDP) formulation. `RSD4` guarantees resource and delay constraints by Lagrangian dual and delay-sensitive queues, respectively. It also efficiently handles partial observability with a memory mechanism enabled by the recurrent neural network (RNN). Moreover, it introduces user-level decomposition and node-level merging to support large-scale multihop scenarios. Extensive experiments on simulated and real-world datasets demonstrate that `RSD4` is robust to system dynamics and partially observable environments and achieves superior performance over existing methods.**

***Index Terms*— Delay-constrained, scheduling, partial observability, deep reinforcement learning.**

I. Introduction

DELAY-CONSTRAINED scheduling has become a crucial problem in guaranteeing a satisfying quality of experience in various domains, including real-time interactive applications such as online games, virtual reality (VR), and cloud computing, due to increasingly rigid user requirements. For instance, express delivery is a typical delay-sensitive scheduling problem, as even a small increase in delivery time can significantly impact customers' perceived ambiguity and reduce satisfaction, as reported by [1]. Delay-constrained scheduling is also critical for data communications [2], video streaming [3], and data center management [4].

Delay-constrained scheduling poses several challenges. Firstly, the scheduler must satisfy latency and resource constraints, while the delay metric depends on the overall dynamics and control of the system across time, and the resource constraint further couples scheduling decisions. Secondly, system dynamics, such as user channels in mobile networks, are hard to trace since distributions of underlying random components can be highly dynamic and correlated. Thirdly, practical scheduling systems usually face a large number of users and have complex network structures, requiring highly scalable solutions for large-scale and multihop scenarios. For example, live video platforms such as YouTube and Instagram have millions of daily active users [5]. Fourthly, practical systems can also suffer from partial observability issues due to sensing noise and hidden correlation. For instance, most IoT devices cannot have perfect knowledge of a dynamic channel environment due to hardware limitations and short sensing time [6], and channel states in network systems can also be challenging to obtain [7]. The universality of partial observability problems demands highly robust algorithms.

Many scheduling algorithms have been proposed in the literature based on different methods, including queueing-based, optimization-based, dynamic programming-based, and Lyapunov control-based approaches. However, these methods have various limitations, such as requiring prior knowledge about system dynamics, suffering from the curse-of-dimensionality, or focusing on stability constraints rather than delay. To address these limitations, this paper proposes a deep reinforcement learning (DRL)-based algorithm, named Recurrent Softmax Delayed Deep Double Deterministic Policy Gradient (`RSD4`), as shown in Figure 1. The proposed algorithm builds on the recurrent deterministic policy gradient and softmax deterministic policy gradient and introduces several novel components to handle the scheduling problem in partially observed settings. Specifically, `RSD4` is an end-to-end method based on a partially observed Markov decision process (POMDP) formulation with a Lagrange dual update. It does not require any prior knowledge of system dynamics and effectively captures hidden system correlation across time slots using a recurrent module. Moreover, `RSD4` employs a softmax operator to improve value estimation during training and enhance robustness in handling complex system dynamics.

Manuscript received 19 March 2023; revised 25 August 2023 and 7 December 2023; accepted 28 December 2023; approved by IEEE/ACM Transactions on Networking Editor I.-H. Hou. Date of publication 9 February 2024; date of current version 18 June 2024. This work was supported by the Technology and Innovation Major Project of the Ministry of Science and Technology of China under Grant 2020AAA0108400 and Grant 2020AAA0108403. *(Corresponding author: Longbo Huang.)*

Pihe Hu, Yu Chen, and Longbo Huang are with the Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University, Beijing 100084, China (e-mail: longbohuang@tsinghua.edu.cn).

Ling Pan is with the Department of Electronic and Computer Engineering, The Hong Kong University of Science and Technology (HKUST), Hong Kong, China (e-mail: penny.ling.pan@gmail.com).

Zhixuan Fang is with the Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University, Beijing 100084, China, and also with the Shanghai Qi Zhi Institute, Shanghai 200232, China (e-mail: zfang@mail.tsinghua.edu.cn).

Fu Xiao is with the School of Computers, Nanjing University of Posts and Telecommunications, Nanjing 210023, China (e-mail: xiaof@njupt.edu.cn).

Digital Object Identifier 10.1109/TNET.2024.3359911

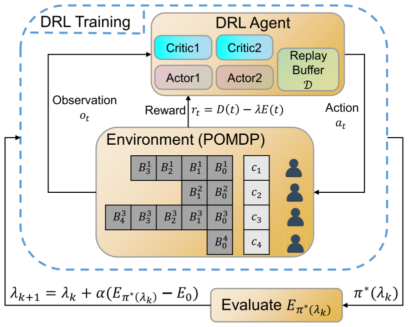

DRL Training
DRL Agent
Critic1
Critic2
Actor1
Actor2
Replay Buffer $\mathcal{D}$
Observation o_t
Reward $r_t = D(t) - \lambda E(t)$
Action a_t
Environment (POMDP)
B_3^1 B_2^1 B_1^1 B_0^1
B_1^2 B_0^2
B_4^3 B_3^3 B_2^3 B_1^3 B_0^3
B_0^4
c_1 c_2 c_3 c_4
$\lambda_{k+1} = \lambda_k + \alpha(E_{\pi^*(\lambda_k)} - E_0)$
Evaluate $E_{\pi^*(\lambda_k)}$
$\pi^*(\lambda_k)$

Fig. 1. RSD4 framework for delay-constrained scheduling.

Finally, it introduces user-level decomposition and node-level merging to support large-scale and multihop scenarios.

In summary, the RSD4 algorithm is a novel DRL-based approach for solving delay-constrained scheduling problems in partially observable systems. It is an end-to-end method that does not require any prior knowledge of the system dynamics, and its base POMDP formulation is general enough to handle various real-world scheduling scenarios. The proposed algorithm is highly scalable, making it suitable for large-scale multihop resource-constrained scenarios, and it introduces several novel components, including the recurrent module and the softmax operator, to improve the value estimation and robustness of the algorithm. The extensive experiments show that RSD4 outperforms classical non-DRL methods and existing DRL benchmarks.

The main contributions of this paper are summarized as follows. (i) We provide a general POMDP framework that is suitable for investigating multi-user delay-constrained scheduling problems with resource constraint, and provides two novel functions for scalability, i.e., user-level decomposition and node-level merging. (ii) We propose a novel DRL-based algorithm, RSD4, which addresses the partial observability issue and achieves robust value learning. It introduces novel components of a unified training approach, a double-branch neural network architecture, and a delayed policy update. (iii) We conduct extensive experiments on both simulated and real-world datasets, demonstrating that RSD4 outperforms existing scheduling methods in various scenarios, especially in partially-observable settings.

II. Related Work

Numerous research efforts have addressed the scheduling problem through various methodologies, encompassing stochastic optimization, combinatorial optimization, and deep reinforcement learning.

Stochastic Optimization: A prominent avenue of research tackles the problem from a stochastic optimization perspective. Among the adopted stochastic optimization techniques, four methodologies stand out: queueing theory-based methods, such as [8] and [9], which pertain to multi-queue systems; convex optimization-based methods, exemplified by [10] and [11], which pertain to wireless network scheduling; dynamic programming (DP)-based control, as found in [12] and [13], aimed at throughput optimal scheduling; and Lyapunov-based optimization, e.g., [14] and [15], for load and energy scheduling. However, these approaches often struggle to explicitly incorporate delay constraints or necessitate precise knowledge of system dynamics, which can be challenging to obtain in real-world scenarios.

Combinatorial Optimization: Another avenue of research delves into the problem from a discrete/combinatorial optimization perspective. For instance, [16] optimizes energy consumption in mobile multicast wireless networks by formulating an equivalent directed Steiner tree problem. Reference [17] optimizes energy consumption with multiple mobile sinks using fuzzy logic in wireless sensor networks. The RCGBSA algorithm is proposed in [18] by incorporating discovered priorities into the extended relative collision graph. Reference [19] addresses the charging scheduling problem in wireless rechargeable sensor networks through solving a reformulated deadline-TSP problem. However, these combinatorial optimization methods often grapple with the curse of dimensionality and struggle to scale to large scenarios.

Deep Reinforcement Learning: In recent years, deep reinforcement learning (DRL) has gained substantial attention in the scheduling domain due to its potential in generalization and scalability. Applications of DRL span various scheduling scenarios, including video streaming [20], Multipath TCP control [2], network reconfigurability [3], and resource-constrained scheduling, as demonstrated in works such as [21] and [22]. Moreover, many scheduling problems adopt partially observable Markov decision process (POMDP) formulations. For instance, scheduling the use of airborne electronic countermeasures in air operations [23], and minimizing the network-wide age of information (AoI) by scheduling end nodes [24], all leverage POMDPs. However, these DRL-based or POMDP-based approaches often struggle to ensure average resource constraints or require a fully observable system state as input. Additionally, they tend to overlook large-scale or multihop systems. In contrast, our proposed RSD4 constitutes a data-driven end-to-end algorithm adept at addressing the partial observability issue under the POMDP formulation. Furthermore, it manages delay and resource constraints through delay-sensitive queues and the Lagrangian dual, respectively. It also embraces user-level decomposition and node-level merging techniques, significantly extending its scalability.

III. Problem Formulation

This section outlines our scheduling problem formulation. Initially, we concentrate on the single-hop setting with an average resource limit across times in Section III-A. Subsequently, we present the corresponding Lagrange dual in Section III-B. Furthermore, we delve into the hard resource limit in each time slot in Section III-C. Finally, we introduce the more general multihop scheduling problem in Section III-D.

A. The Scheduling Problem

We consider a scheduling problem (Figure 2), where time is divided into discrete slots $t \in 0, 1, 2, \ldots$, and the scheduler

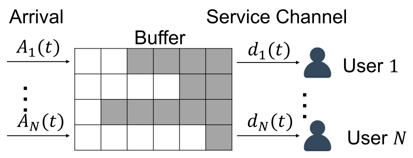


Fig. 2. A delay-constrained single-hop network. Jobs arrive at the server and need to be delivered before the deadline.

receives job arrivals, such as data packets in a network or parcels in a delivery station, at the beginning of each time slot. The number of job arrivals for user i at time slot t is denoted as $A_i(t)$, and we denote $\boldsymbol{A}(t) = [A_1(t), \dots, A_N(t)]$. Each job for user i has a strict delay constraint τ_i, which means that the job must be served within τ_i slots upon its arrival, or it will become outdated and be discarded.

The buffer model: Jobs arriving at the system are initially stored in a buffer, which is modeled by a set of delay-sensitive queues. Specifically, the buffer comprises N separate delay-sensitive queues, one for each user, and each queue has an infinite size. The state of queue i at time slot t is denoted by $\boldsymbol{B}_i(t) = [B_i^0(t), B_i^1(t), \dots, B_i^{\tau_i}(t)]$, where $B_i^\tau(t)$ represents the number of jobs for user i that have a remaining time of τ timeslots until expiration for $1 \le \tau \le \tau_i$.

The scheduling and service model: At every time slot t, the scheduler makes decisions on the resources allocated to jobs in the buffer. For each user i, the decision is denoted as $\boldsymbol{e}_i(t) = [e_i^0(t), e_i^1(t), \dots, e_i^{\tau_i}(t)]$, where $e_i^\tau(t) \in [0, e_{\max}]$ represents the amount of resource allocated to serve each job in queue i with a deadline of τ. Each scheduled job is then passed through a service channel, whose condition is random and represented by $c_i(t)$ for user i at time slot t. The full channel conditions at time slot t are denoted as $\boldsymbol{c}(t) = [c_1(t), c_2(t), \dots, c_N(t)]$. The probability of successful service for a user i's job with allocated resource e and channel condition c is denoted as $P_i(e, c)$. Here, $e = 0$ means that the job will not be served, and $P_i(0, c) = 0$. Additionally, $P_i(e, \cdot) > 0$ for all $e > 0$. If a job is scheduled but fails in service, it remains in the buffer if it is not outdated. The instantaneous resource consumption at time slot t is denoted as $E(t) = \sum_{i=1}^N \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)$, and the average resource consumption is $\overline{E} = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^T E(t)$.

The system objective: For each user i, the number of jobs that are successfully served at time slot t is denoted as $d_i(t)$. A weight β_i is assigned to each user, and the weighted instantaneous throughput is defined as $D(t) = \sum_{i=1}^N \beta_i d_i(t)$. The objective of the scheduler is to maximize the weighted average throughput, defined as $\overline{D} = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^T D(t)$, subject to the average resource consumption limit, i.e.,[1]

$$\mathcal{P}: \max_{\boldsymbol{e}_i(t): 1\le i\le N, 1\le t\le T} \quad \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^T \sum_{i=1}^N \beta_i d_i(t)$$
$$\text{s.t.} \quad \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^T \sum_{i=1}^N \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t) \le E_0 \qquad (1)$$

where E_0 is the average resource consumption limit. We denote the optimal value of problem $\mathcal{P}$ by $\mathcal{T}^*$.

[1]We assume w.l.o.g. that all corresponding limits exist. The results can be generalized with $\limsup$ and $\liminf$ definitions otherwise.

Note that while our problem formulation is based on that in [12], our aim is to devise practical and scalable scheduling solutions that can be applied in real-world scenarios. In doing so, we avoid making assumptions about the ergodicity of system dynamics and instead allow for randomness that may be correlated with hidden factors. We start by not enforcing the hard constraints on resource allocation in our approach. Then, in Section III-C, we demonstrate that our framework can also incorporate hard constraints if needed.

B. Lagrange Dual

We define a Lagrangian function to handle the average resource budget constraint in problem $\mathcal{P}$, as follows:

$$\mathcal{L}(\pi, \lambda) = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^T \sum_{i=1}^N \left[\beta_i d_i(t) - \lambda \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i^\top(t)\right] + \lambda E_0 \qquad (2)$$

where π is the control policy and λ is the Lagrange multiplier. We denote by $g(\lambda)$ the Lagrange dual function for a given Lagrange multiplier λ, given by:

$$g(\lambda) = \max_\pi \mathcal{L}(\pi, \lambda) = \mathcal{L}(\pi^*(\lambda), \lambda), \qquad (3)$$

where the maximizer is denoted as $\pi^*(\lambda)$. Using Lemma 3 in [12], the optimal timely throughput $\mathcal{T}^*$ equals the optimal value of the dual problem, i.e., $\mathcal{T}^* = \min_{\lambda\ge 0} g(\lambda) = g(\lambda^*)$, where λ^* is the optimal Lagrange multiplier.

To obtain the optimal policy $\pi^*(\lambda^*) = \arg\max_\pi \mathcal{L}(\pi, \lambda^*)$, it is necessary to first find the optimal Lagrangian multiplier λ^*. Here, we denote the consumed resource under policy π in time slot t as $E_\pi(t)$. According to Danskin's Theorem in [25], the derivative of the dual function $g(\lambda)$ can be expressed as $g'(\lambda) = \frac{\partial \mathcal{L}(\pi^*(\lambda),\lambda)}{\partial \lambda} = E_0 - E_{\pi^*(\lambda)}$, where $E_{\pi^*(\lambda)} = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^T E_{\pi^*(\lambda)}(t)$ denotes the average resource consumption under the optimal policy $\pi^*(\lambda)$. Therefore, the optimal policy $\pi^*(\lambda^*)$ can be obtained by recovering the dual function $g(\lambda)$ for some λ and using gradient descent to find the optimal $\lambda^* > 0$ that minimizes $g(\lambda)$.

In practical systems, finding the optimal policy $\pi^*(\lambda)$ for a given λ is challenging due to several reasons. First, system dynamics are difficult to trace because distributions can be highly dynamic and correlated in many scenarios. Second, the system can only be partially observed due to sensing limitations, noise, and other hidden factors. Finally, practical scheduling systems usually involve a large number of users and a complex multihop topology, which require highly scalable solutions. Previous works [12], [13] have used dynamic programming to find the maximizer $\pi^*(\lambda)$. However, this approach requires prior knowledge of system dynamics and suffers from the curse-of-dimensionality in large systems, and may not be directly applicable to partially observable systems. These challenges have motivated us to design a DRL-based framework with partially observable Markov decision process formulation in Section IV.

C. Hard Resource Limit

We consider $\mathcal{P}$ with a hard resource limit of E_h in each time slot. To incorporate this limit in our POMDP formulation mentioned in Section IV-A, we convert the constrained maximization problem to an unconstrained one. Hard resource constraints, such as the transmitting antennas in wireless communication systems, are common in realistic scheduling problems. Therefore, we introduce the following problem:

$$\overline{\mathcal{P}}: \max_{\boldsymbol{e}_i(t):1\le i\le N,1\le t\le T} \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N}\beta_i d_i(t)$$
$$\text{s.t. } \lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{N}\boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)\le E_0$$
$$\sum_{i=1}^{N}\boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)\le E_h, \forall 1\le t\le T \quad (4)$$

Here E_h is the hard resource limit of the server. The Lagrangian function for problem $\overline{\mathcal{P}}$ can be obtained similarly to Eq. (2), except for further considering the hard resource limit $\sum_{i=1}^{N}\boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)\le E_h$ for any $1\le t\le T$, i.e.,

$$\overline{g}(\lambda)=\max_{\pi:\sum_{i=1}^{N}\boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)\le E_h,\forall 1\le t\le T}\mathcal{L}(\pi,\lambda). \quad (5)$$

As discussed in Section III-B, obtaining the optimal policy involves maximizing the Lagrangian function, which is non-trivial due to the presence of the hard resource limit. To tackle this challenge, we aim to transform the constrained maximization problem into an unconstrained one, to facilitate the design of DRL-based solutions. To this end, we design the following auxiliary Lagrange function:

$$\widetilde{\mathcal{L}}(\pi,\lambda)=\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\mathbb{1}\left\{\sum_{i=1}^{N}\boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)\le E_h\right\}$$
$$\cdot\sum_{i=1}^{N}\left[\beta_i d_i(t)-\lambda\boldsymbol{e}_i^\top(t)\boldsymbol{B}_i^\top(t)\right]+\lambda E_0, \quad (6)$$

Here we use an indicator function to ensure that the resource limit is satisfied for all time steps $1\le t\le T$. The accompanying auxiliary Lagrange dual function is:

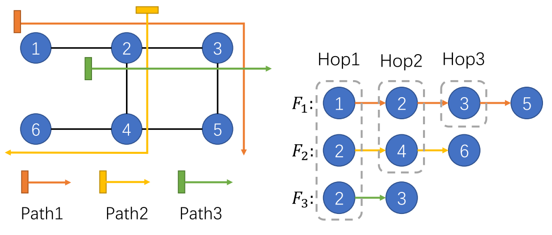
$$\widetilde{g}(\lambda)=\max_{\pi}\widetilde{\mathcal{L}}(\pi,\lambda)=\widetilde{\mathcal{L}}(\widetilde{\pi}^*(\lambda),\lambda), \quad (7)$$

where $\widetilde{\pi}^*(\lambda)$ denotes the maximizer of the auxiliary Lagrange function under the multiplier λ. By introducing the auxiliary Lagrange function and dual function, we can reformulate the constrained optimization problem $\overline{\mathcal{P}}$ into an unconstrained one, thereby facilitating the computation of the optimal policy. Lemma 1 below establishes the equivalence of the reformulation by revealing the relationship between the Lagrange dual function $\overline{g}(\lambda)$ and the auxiliary Lagrange dual function $\widetilde{g}(\lambda)$.

Lemma 1: For any $\lambda>0$, if $\widetilde{\pi}^*(\lambda)$ is the maximizer of $\widetilde{g}(\lambda)$, then the policy obtained by truncation operation (Appendix A) of $\widetilde{\pi}^*(\lambda)$,[2] denoted by $\overline{\pi}^*(\lambda)$, is a maximizer of $\overline{g}(\lambda)$.

Proof: Please refer to Appendix A. □

[2]The truncated policy $\overline{\pi}^*(\lambda)$ means the policy whose output is the truncated action of the output of $\widetilde{\pi}^*(\lambda)$, under the same system state.



(a) A multihop network. (b) Three aligned flows

Fig. 3. (a) A multihop network with three paths in different colours, each supporting one flow. (b) The flows are aligned with starting nodes.

Remark 2: There is no hard resource limit involved in maximizing the auxiliary Lagrange function $\widetilde{\mathcal{L}}$. Consequently, by maximizing the auxiliary Lagrange function $\widetilde{\mathcal{L}}$ for a given $\lambda>0$, we can derive the optimal policy $\overline{\pi}^*(\lambda)$ for the original Lagrange function $\overline{\mathcal{L}}$, which does include a hard resource limit, by truncating $\widetilde{\pi}^*(\lambda)$. The rationale behind transforming the constrained maximization problem into an unconstrained one is to enable the incorporation of the hard resource limit into our POMDP formulation, as shown in Section IV-A.

D. Multihop Networks

In a multihop network, each flow traverses multiple hops, and the scheduler must determine which flows to serve and how many resources to allocate at each intermediate node. Our multihop network consists of N flows (or N users), denoted by $F_1,F_2,\dots,F_N$, and K nodes $1,2,\dots,K$ involved in scheduling. The path length of flow i is denoted as J_i, and the maximum hop length of all flows is denoted as $J=\max_{1\le i\le N}J_i$. We represent flow i's topology using a matrix $\mathbf{H}^{(i)}=(h_{jk}^{(i)})\in\mathbb{R}^{J\times K}$, where $h_{jk}^{(i)}=1$ if flow i's j-th hop is node k, otherwise, $h_{jk}^{(i)}=0$. The number of job arrivals for flow i at time slot t is denoted as $A_i(t)$, and each job for flow i has a strict delay constraint τ_i. Each node has an average resource constraint E_i. As an example, we consider the multihop network with three paths in Figure 3a. Three flows pass through three paths, namely $F_1=1\to2\to3\to5$, $F_2=2\to4\to6$, and $F_3=2\to3$.

The multihop scheduling problem can be considered as the aggregation of multiple single-hop scheduling problems, with the specific buffer model, scheduling and service model, and system objective for multihop cases as follows.

The aggregated buffer model: In a multihop network, the buffers are modelled as a set of delay-sensitive queues similar to the single-hop case. However, since jobs in a multihop network need to traverse multiple hops before reaching their destinations, the buffer model needs to account for the flow of jobs through the entire network. To better represent the system state, we propose an aggregated buffer model that aligns flows with their starting nodes, as shown in Figure 3b. For flow i with a path length of J_i, we denote the buffer state of flow i at j-th hop $(1\le j\le J_i)$ as $\boldsymbol{B}_i^{(j)}(t)=[B_i^{(j,0)}(t),B_i^{(j,1)}(t),\dots,B_i^{(j,\tau_i-j+1)}(t)]$, where $B_i^{(j,\tau)}(t)$ represents the number of jobs in flow i at j-th hop with τ remaining time slots until expiration, for $0\le\tau\le\tau_i-j+1$. We set $\boldsymbol{B}_i^{(j)}(t)=\mathbf{0}$ for $J_i<j\le J$ to denote the absence of flow i in those hops. In this manner, there

are J aggregated buffers, i.e., $\mathbf{B}^{(1)}(t), \mathbf{B}^{(2)}(t), \ldots, \mathbf{B}^{(J)}(t)$, where $\mathbf{B}^{(j)}(t) = [\boldsymbol{B}_1^{(j)}(t), \boldsymbol{B}_2^{(j)}(t), \ldots, \boldsymbol{B}_N^{(j)}(t)]$ denotes the j-th aggregated buffer for $1 \leq j \leq J$.

The multihop scheduling and service model: At every time slot t, the scheduler must determine which flows to serve and how many resources to allocate at each node. The allocated resources for flow i are given by $\mathbf{e}_i(t) = [\boldsymbol{e}_i^{(1)}(t), \boldsymbol{e}_i^{(2)}(t), \ldots, \boldsymbol{e}_i^{(J_i)}(t)]$, where $\boldsymbol{e}_i^{(j)}(t)$ represents the resources allocated to flow i's j-th hop for $1 \leq j \leq J_i$. Thus, the scheduling decision for all flows (users) at time slot t is denoted as $a_t = [\mathbf{e}_1(t), \mathbf{e}_2(t), \ldots, \mathbf{e}_N(t)]$. Each scheduled job needs to pass through service channels in its flow path. We denote the service channel condition between node i and j at time slot t as $c_{ij}(t)$. For a job of flow i with allocated resources e and channel condition c, the probability of successful service is denoted as $P_i(e,c)$. The instantaneous resource consumption of node k at time slot t is denoted as $E^{(k)}(t) = \sum_{i=1}^N \sum_{j=1}^{J_i} h_{jk}^{(i)} {\boldsymbol{e}_i^{(j)}}^\top(t) \boldsymbol{B}_i^{(j)}(t)$, and the average resource consumption of node k is denoted as $\overline{E}^{(k)} = \lim T \rightarrow \infty \frac{1}{T} \sum_{t=1}^T E^{(k)}(t)$.

The multihop system objective: The number of successfully served jobs at time slot t for flow i is denoted as $d_i(t)$. Each flow is also given a weight β_i, and the weighted instantaneous throughput is denoted as $D(t) = \sum_{i=1}^N \beta_i d_i(t)$. The objective of the scheduler is to maximize the weighted average throughput, defined as $\overline{D} = \lim_{T\rightarrow\infty} \frac{1}{T}\sum_{t=1}^T D(t)$, subject to the average resource consumption limit, i.e.,

$$\begin{aligned} \mathcal{P}_m: \max_{\boldsymbol{e}_i(t):1\leq i\leq N, 1\leq t\leq T} \quad & \lim_{T\rightarrow\infty} \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^N \beta_i d_i(t) \\ \text{s.t.} \quad & \lim_{T\rightarrow\infty} \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^N \sum_{j=1}^{J_i} h_{jk}^{(i)} {\boldsymbol{e}_i^{(j)}}^\top(t) \cdot \\ & \boldsymbol{B}_i^{(j)}(t) \leq E_0^{(k)}, \forall 1 \leq k \leq K \end{aligned} \tag{8}$$

where $E_0^{(k)}$ is the average resource limit of node k.

Lagrange Dual for Multihop cases: The Lagrangian function for multihop problem $\mathcal{P}_m$ is given as

$$\begin{aligned} &\mathcal{L}_m(\pi, \lambda) \\ &= \lim_{T\rightarrow\infty} \frac{1}{T} \sum_{t=1}^T \sum_{i=1}^N \Big[\beta_i d_i(t) \\ &\quad - \sum_{k=1}^K \lambda_k \sum_{j=1}^{J_i} h_{jk}^{(i)} {\boldsymbol{e}_i^{(j)}}^\top(t) \boldsymbol{B}_i^{(j)}(t)\Big] + \sum_{k=1}^K \lambda_k \lambda E_0^{(k)}, \end{aligned} \tag{9}$$

where $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_K]$ contains multiple Lagrange multipliers. Besides, the Lagrange dual function for fixed Lagrange multiplier $\boldsymbol{\lambda}$ can be given as:

$$g_m(\boldsymbol{\lambda}) = \max_\pi \mathcal{L}_m(\pi, \boldsymbol{\lambda}) = \mathcal{L}_m(\pi_m^*(\boldsymbol{\lambda}), \boldsymbol{\lambda}), \tag{10}$$

Thus, the optimal policy $\pi_m^*(\boldsymbol{\lambda}^*)$ can also be obtained by recovering the dual function $g_m(\boldsymbol{\lambda})$ for some $\boldsymbol{\lambda}$ and taking gradient descent to find the optimal $\boldsymbol{\lambda}^*$.

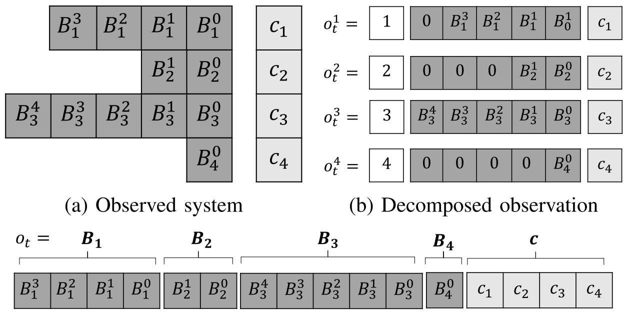


Fig. 4. Observations of a four-user system (time index omitted).

IV. Overall Framework

This section presents the overall framework of our solution to the scheduling problem. We begin by introducing the POMDP formulation in Section IV-A, followed by our approach to achieving scalability through user-level decomposition and node-level merging in Sections IV-B and IV-C.

A. The POMDP Formulation

We present the POMDP formulation for the scheduling problem, which enables the application of RSD4 to find the optimal policy $\pi(\lambda)$. The POMDP is denoted by $\mathcal{M} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, r, \mathbb{P}, \gamma \rangle$, where $\mathcal{S}$ represents the state space, $\mathcal{O}$ denotes the observation space, $\mathcal{A}$ is the action space, r is the reward function, $\mathbb{P}$ denotes the transition matrix, and γ stands for the discount factor.

State and Observation: The overall system state s_t includes $\boldsymbol{A}(t)$, $\boldsymbol{B}_1(t)$, ..., $\boldsymbol{B}_N(t)$, $\boldsymbol{c}(t)$, and other information related to the underlying MDP that is unobservable by the scheduler, such as random hyperparameters of successful transmission probability or the mobile user's position that affects traffic arrival and channel. Since partial observable scenarios are common in scheduling problems, we consider s_t to be partially observed. This implies that the actual observation o_t is a subset of s_t, and the exact form of o_t depends on the environment settings. For example, Figure 4a shows a partially-observed system of a four-user case, where the observation $o_t = [\boldsymbol{B}_1(t), \ldots, \boldsymbol{B}_N(t), \boldsymbol{c}(t)]$ only includes the buffer with delay-sensitive queues and channel states.

Action and Reward: At time slot t, the action is denoted by $a_t = [\boldsymbol{e}_1(t), \ldots, \boldsymbol{e}_N(t)]$, and the reward is

$$r_t = D(t) - \lambda E(t), \tag{11}$$

which is the instantaneous weighted throughput $D(t)$ minus the resource consumption $E(t)$ weighted by the λ.

Learning Objective: Under POMDP, an optimal agent needs to access the entire history $h_t = (o_1, a_1, o_2, a_2, \ldots, a_{t-1}, o_t)$ and learn a deterministic policy $\pi(\cdot;\phi)$ parameterized by ϕ, which maps from the history to the action space, with the objective of maximizing the expected long-term rewards

$$J(\pi(\cdot;\phi)) = \mathbb{E}\left[\sum_{t=0}^T \gamma^t r(s_t, a_t) \mid s_0, a_0, \pi(\cdot;\phi)\right].$$

Remark 3: The reward setting in (11) implies that when $\gamma = 1$, the cumulative discounted reward is $R = \sum_{t=1}^{T} \gamma^t r_t = T\mathcal{L}(\pi, \lambda) - \lambda T E_0$. This differs from the Lagrangian function value in (2) by $-\lambda T E_0$. Therefore, an algorithm maximizing the expected rewards $J(\pi(\cdot;\phi))$ is also a maximizer for the Lagrangian function, which is the objective of RSD4 *as detailed in Section V. Furthermore, if we consider the hard resource limit, it is sufficient to set the reward as*

$$r_t = \mathbb{1}\left\{\sum_{i=1}^{N} \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t) \le E_h\right\} \cdot \sum_{i=1}^{N}[\beta_i d_i(t) - \lambda \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i^\top(t)]$$
$$= \mathbb{1}\{E(t) \le E_h\}[D(t) - \lambda E(t)] \tag{12}$$

according to Remark 2 by the auxiliary Lagrangian function.

B. User-Level Decomposition

In a multi-user scheduling system, the inclusion of buffer information in the system state, whose size scales with the number of users, poses a considerable challenge to learning, particularly in large-scale systems. This challenge stems from the necessity to train neural networks with an increasing number of hyperparameters due to the larger input dimension. As a result, the scalability of DRL-based methods is significantly limited. To address this limitation, we introduce a *user-level decomposition* technique that empowers RSD4 to devise effective policies using compact neural networks, even when dealing with large-scale scenarios.

User-level decomposition transforms the original maximization problem of Eq. (2) into a series of decomposed user-specific sub-problems. This approach involves defining the user-level decomposed Lagrangian function for user i as:

$$\mathcal{L}_i(\pi_i, \lambda_i) = \lim_{T\to\infty} \frac{1}{T}\left[\sum_{t=1}^{T} \beta_i d_i(t) - \lambda_i \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i^\top(t)\right], \tag{13}$$

Here, π_i represents the decomposed policy for scheduling user i's jobs in the buffer. The interpretation of $\mathcal{L}_i$ is as follows: the packet accumulates a payment of λ_i per unit energy used for transmission by user i, while simultaneously accruing a reward of β_i upon reaching its destination before the expiration of its deadline.

We denote the optimal policy for user i's sub-problem as $\pi_i^*(\lambda_i)$, where λ_i serves as the multiplier. After maximizing the decomposed Lagrangian function (13) for each user with a fixed $\lambda_i = \lambda$, we can obtain the optimal policy $\pi^*(\lambda)$ by aggregating each user's individual optimal scheduling policy $\pi_i^*(\lambda_i)$, which is proved in Lemma 4.

Lemma 4: For a fixed λ, let $\widehat{\pi}(\lambda)$ denote the aggregated policy of $\{\pi_1^*(\lambda), \pi_2^*(\lambda), \ldots, \pi_N^*(\lambda)\}$. This aggregated policy operates on an observation $o = [o^{(1)}, o^{(2)}, \ldots, o^{(N)}]$ and yields an output $\widehat{\pi}(\lambda)(o) = [\pi^*(\lambda)(o^{(1)}), \pi^*(\lambda)(o^{(2)}), \ldots, \pi^*(\lambda)(o^{(N)})]$. The policy $\widehat{\pi}(\lambda)$ is then the optimal policy that maximizes the Lagrangian function in Eq. (2).

Proof: please refer to Appendix B. □

Furthermore, our approach to user-level decomposition differs from the packet-level decomposition method employed in previous works such as [12] and [13]. The distinction arises from the observation that packet-level decomposition generates numerous packet-specific sub-problems, thereby introducing additional instability that hinders the neural network's ability to learn the optimal policy effectively. Consequently, our user-level decomposition technique empowers RSD4 to identify the optimal policy using compact neural networks that maintain stability during training, even in large-scale scenarios, which significantly enhances the scalability of our approach.

Decomposed POMDP: Based on the above intuition, we decompose the POMDP into user-level subproblems, where for each user i, the action is $a_t^{(i)} = \boldsymbol{e}_i(t)$, and the reward becomes $r_t^{(i)} = \beta_i d_i(t) - \lambda \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)$ at time slot t. The observation is also decomposed as well, Figure 4b presents $o_t^{(i)} = [i, \boldsymbol{B}_i(t), \boldsymbol{c}_i(t)]$ for a four-user case, where the first index for user i's sub-problem is required for algorithm training (explained next). This is a key step in RSD4. As we will see in Section VI-D, without decomposition, RSD4 and other existing DRL algorithms can fail due to inadequate state representation under large state scenarios.

Unified Training: After decomposition, the state space is compressed, resulting in N different sub-POMDPs, each with different dynamics for different users. Training N different DRL agents would result in a linear growth of computation power, which is not feasible for large-scale systems. To address this issue, we propose the method of unified training. In this method, we train different samples from different sub-environments together, using an extra user index i as the identifier for samples from the user i's sub-problem, as shown in Figure 4b. Consequently, the dimension of training samples remains the same regardless of the system scale, which achieves better scalability.

Remark 5: With observation decomposition and unified training, a single system observation o_t is decomposed into N separate sate $o_t^1, o_t^2, \ldots, o_t^N$, such that one scheduling decision in the original environment creates N samples for the replay buffer. Consequently, it does not require heavy parallel computation and greatly enriches the abundance of the replay buffer, such that common knowledge across different users' sub-POMDPs is learned efficiently. With PODMP decomposition, the number of neural network parameters also remains small even for large-scale systems, because the dimension of observation remains unchanged after decomposition, which makes our framework highly scalable.

C. Node-Level Merging

We introduce a technique called *node-level merging* to address the complexity of multihop scheduling problems in POMDP formulation. The main idea of node-level merging is to augment the state, observation, action, and reward in the POMDP formulation by grouping flows with same hop numbers. Figure 5 illustrates an example of node-level merging with the detailed procedure below.

State and Observation: The system state is obtained by merging buffer and channel states at different hops. Given that jobs in a multihop flow differ only in their node position and remaining time until expiration, we aggregate the buffer states at each node to encode the system state. Specifically,

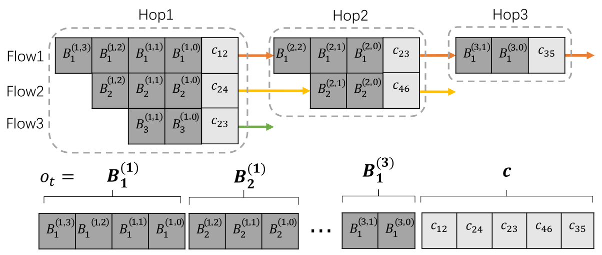


Fig. 5. Node-level Merging of a three-flow case. Flow i' node j is associated with a buffer $\boldsymbol{B}_j^i$. Flows are aligned with starting nodes.

for flow i with a path length of J_i, we have denoted the buffer state of flow i at j-th hop ($1 \leq j \leq J_i$) as $\boldsymbol{B}_i^{(j)}(t) = [B_i^{(j,0)}(t), B_i^{(j,1)}(t), \ldots, Bi^{(j,\tau_i-j+1)}(t)]$. The idea of node-level merging is to set the overall observation o_t to be $[\boldsymbol{B}_1^{(1)}(t), \ldots, \boldsymbol{B}_N^{(1)}(t), \ldots, \boldsymbol{B}_1^{(J)}, \ldots, \boldsymbol{B}_N^{(J)}(t), \boldsymbol{c}(t)]$, along with potentially other environmental factors. Figure 5 illustrates a case where $o_t = [\boldsymbol{B}_1^{(1)}(t), \ldots, \boldsymbol{B}_N^{(1)}(t), \ldots, \boldsymbol{B}_1^{(J)}, \ldots, \boldsymbol{B}_N^{(J)}(t), \boldsymbol{c}(t)]$.

Action: The action is similarly obtained by merging actions at different nodes, i.e., $a_t = [\boldsymbol{e}_1^{(1)}(t), \ldots, \boldsymbol{e}_N^{(1)}(t), \boldsymbol{e}_1^{(J)}(t), \ldots, \boldsymbol{e}_N^{(J)}(t)]$, where $\boldsymbol{e}_i^{(j)}(t)$ represent the resource allocated to user i's j-th node for $1 \leq j \leq J_i$.

Reward: The reward is set to $r_t = D(t) - \sum_{k=1}^{M} \lambda_k E^{(k)}(t)$, where λ_k and $E^{(k)}(t)$ correspond to the Lagrangian multiplier and resource consumption at node k for $1 \leq k \leq K$.

A common approach for multihop scheduling is to address the scheduling problem in each node separately. However, in delay-constrained multihop networks, scheduling each node separately cannot satisfy the hard delay constraint. Multi-agent architecture [26], [27] has been adopted for multihop networks but with more computation resource requirements and can be harder to train. Node-level merging provides a novel way of concatenating buffers of different hops in flows together, making its training similar to that for a single-hop scheduling problem except with a higher input dimension. This avoids training multiple agents for multihop networks, and efficiently reduces the complexity in training. It allows RSD4 to perform well and outperforms other methods, even when the system scale is significantly increased in multihop networks.

V. Proposed Method: RSD4

We present our novel algorithm, called RSD4, which stands for Recurrent Softmax Delayed Deep Double Deterministic Policy Gradient, in Algorithm 1. This algorithm builds upon the recurrent deterministic policy gradient [28] and softmax deterministic policy gradient [29] and introduces several novel components for handling the scheduling problem in partially observed settings. RSD4 is a model-free deep reinforcement learning algorithm that utilizes the actor-critic framework [30]. It is designed to handle partial observability issues using the memory mechanism enabled by recurrent neural networks (RNNs) and resolves the overestimation problem in existing recurrent DRL methods using the softmax operator. As a result, RSD4 has advantages over both recurrent deterministic policy gradient [28] and softmax deterministic policy gradient [29]. Additionally, RSD4 adopts a double-branch architecture from [31] to better utilize the memory mechanism of the RNN and implements a delayed policy update frequency to further reduce the variance in value estimates. Overall, RSD4 is a powerful algorithm that can handle partial observability issues and provide accurate value estimates for efficient scheduling in complex systems.

Initially, RSD4 makes use of the state-action function $Q(h_t, a_t; \theta)$ parameterized by θ, which is defined as

$$Q(h_t, a_t; \theta) = \mathbb{E}_{s_t, a_t, \ldots, s_{t+T}, a_{t+T} | h_t, \pi(\cdot;\phi)} \left[\sum_{i=0}^{T} \gamma^i r(s_{t+i}, a_{t+i}) \right], \quad (14)$$

where the expectation is taken with respect to the conditional probability $p(s_t, a_t, \ldots, s_{t+T}, a_{t+T} | h_t, \pi(\cdot;\phi))$ of the trajectory distribution associated with history h_t and the policy $\pi(\cdot;\phi))$. RSD4 initializes double critic networks and double actor networks, where critic networks $Q(h_t, a_t; \theta)$ estimate the value of state-action pairs, and actor networks $\pi(\cdot;\phi)$ are responsible for outputting control actions.

A. The Training Algorithm

Recurrent Network Architecture: Despite the success of RL in solving a number of challenging tasks, the state-of-the-art RL algorithms such as TD3 [32] have limitations in solving fully-observable tasks. As a result, they may fail when faced with partially observable tasks such as the problem considered in this work. To address this problem, compared to prior non-recurrent policy gradient methods, RSD4 incorporates recurrency in designing the architecture for neural networks instead of simply using feedforward networks in the policy update. This strengthens the memory capability of RSD4 and enables it to learn hidden factors or temporal correlations of the system dynamics. We then update the policy by RDPG [28]:

$$\nabla_\phi J(\pi(\cdot;\phi)) = \mathbb{E}_{h_t} \left[\sum_t \nabla_\phi (\pi(h_t;\phi)) \nabla_a Q(h_t, a; \theta) \Big| a = \pi(h_t;\phi) \right]. \quad (15)$$

To compute RDPG, a sequence of episodes is stored in the replay buffer $\mathcal{D}$ as training samples (line 14). RSD4 computes target values $(y_1, y_2, \ldots, y_T)$ for each sampled episode using the recurrent networks in lines 17-23. Critics and actors are updated recurrently, as shown in lines 24 and 26.

Softmax Double Learning: Having a good estimate of the value function is critical for RL agents to achieve good performance [29]. We therefore propose to incorporate the softmax operator for more accurate value function estimation in different scenarios. In lines 21–22, we compute the Q-function by softmax operator by:

$$\mathrm{softmax}_\beta(Q(h, \cdot)) = \frac{\mathbb{E}_{a\sim p}\left[\frac{\exp(\beta Q(h,a;\theta))Q(h,a;\theta)}{p(a)}\right]}{\mathbb{E}_{a\sim p}\left[\frac{\exp(\beta Q(h,a;\theta))}{p(a)}\right]} \quad (16)$$

where β denotes the inverse temperature and p is the sampling distribution. The target value for critic Q_i is given by $y_i = r + \gamma\, \mathrm{softmax}_\beta(\hat{Q}(h, \cdot))$ in line 22, where $\hat{Q}(h, \cdot)$ denotes the

Algorithm 1 RSD4 With Decomposition

```
Require: $\lambda_0$, resource limit $E_0$, learning rate $\alpha$, Precision $\delta$, episode number $M$, episode length $T$, batch size $b$, target update rate $\tau$, inverse temperature $\beta$, and sampling distribution $p$.
1:  $\lambda_1 \leftarrow \lambda_0, \lambda_0 \leftarrow 0, k \leftarrow 1$
2:  while $|\lambda_k - \lambda_{k-1}| > \delta$ do
3:    Initialize $N$ learning environments with $r_t^{(i)} = \beta_i d_i(t) - \lambda \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)$ for $i$-th environment.
4:    Initialize critic networks $Q_1$, $Q_2$, and actor networks $\pi_1$, $\pi_2$ with random parameters $\theta_1, \theta_2, \phi_1, \phi_2$. Initialize target network $\theta_1^-, \theta_2^-, \phi_1^-, \phi_2^- \leftarrow \theta_1, \theta_2, \phi_1, \phi_2$.
5:    Initialize replay buffer $\mathcal{D}$.
6:    for episodes $= 1$ to $M$// Episodic interaction do
7:      for $t = 1$ to $T$ do
8:        for $i = 1$ to $N$ do
9:          Receive sub observation $o_t^i$
10:         $h_t^i \leftarrow h_{t-1}^i, a_{t-1}^i, o_t^i$
11:         Select action $a_t^i$ based on $\pi_1$ and $\pi_2$.
12:       end for
13:     end for
14:     Store $(o_1^i, a_1^i, r_1^i, \dots, o_T^i, a_T^i, r_T^i)$ in $\mathcal{D}$ for $i = 1$ to $N$.
15:     for $i = 1, 2$ // Double learning do
16:       Randomly sample a batch of $b$ episodes: $\mathcal{B} = \{(o_1, a_1, r_1, \dots, o_T, a_T, r_T)\}$ from $\mathcal{D}$.
17:       for $t = 1$ to $T$ // Recurrent softmax learning do
18:         Sample $K$ noise $\epsilon \sim \mathcal{N}(0, \sigma')$
19:         $\hat{a}_t \leftarrow \pi_i(h_t; \phi_i^-) + clip(\epsilon, -c, c)$
20:         $\hat{Q}(h_t, \hat{a}_t) \leftarrow \min_{j=1,2}(Q_j(h_t, \hat{a}_t; \theta_j^-))$
21:         Compute $\mathrm{softmax}_\beta(\hat{Q}(h_t, \cdot))$ by Eq. (16)
22:         $y_t \leftarrow r + \gamma(1-d)\,\mathrm{softmax}_\beta(\hat{Q}(h_t, \cdot))$, where done flag $d = 1$ if $t = T$ else $d = 0$.
23:       end for
24:       Update $\theta_i$ according to Bellman loss: $\frac{1}{N}\sum_{h\in\mathcal{B}}\sum_t \left(Q_i\left(h_t, a; \theta_i\right) - y_i\right)^2$
25:       if episodes mod $d = 0$ // Delayed update then
26:         Update actor $\phi_i$ by recurrent policy gradient:
            $$\nabla_{\phi_i} J(\phi_i) = \frac{1}{N}\sum_{h\in\mathcal{B}}\sum_t \Big[\nabla_{\phi_i}\left(\pi(h_t;\phi_i)\right) \nabla_a Q_i(h_t, a; \theta_i)|_{a=\pi(h_t;\phi_i)}\Big]$$
27:         Update target networks: $\theta_i^- \leftarrow \tau\theta_i + (1-\tau)\theta_i^-$, $\phi_i^- \leftarrow \tau\phi_i + (1-\tau)\phi_i^-$
28:       end if
29:     end for
30:   end for
31:   Obtain policy $\pi_k$ and evaluate $E_{\pi_k}$.
32:   $\lambda_k = \lambda_k + \alpha(E_{\pi_k} - E_0)$ // Gradient update
33:   $\lambda_{k-1} \leftarrow \lambda_k$
34: end while
35: Output $\pi_k$
```

value estimation function in line 20. RSD4 further adopts a delayed policy update mechanism from [32] (line 25) to avoid training divergence due to frequent updates of the policy. Thus, the policy network is updated at a lower frequency than the value network to minimize error before introducing a policy update, with the similar goal of making the scheduling policy more robust in various systems.

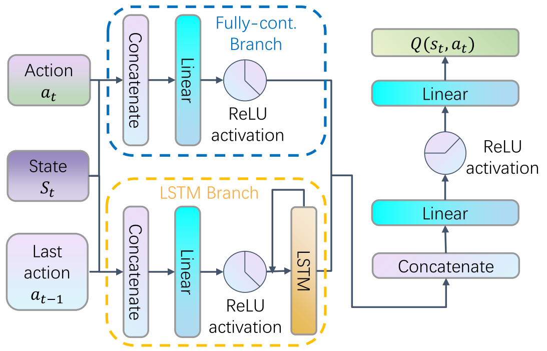


Fig. 6. The architecture of the critic network. There are double parallel branches of the LSTM branch and the fully-connected branch, which are later concatenated together by a fully-connected layer to output Q value.

B. Network Architecture

RSD4 uses double actor-networks and double critic-networks. The architecture of the critic network is shown in Figure 6, and actor networks are similar, with the difference being removing the action a_t in the input and changing the output value $Q(s_t, a_t)$ to action a_t. The recurrent layers build upon Long-Short-Term-Memory (LSTM) [33] to perform RDPG in Eq. (15), and there are two parallel branches, i.e., the fully connected branch and the LSTM branch. This architecture is firstly proposed in [31] and is effective for our scheduling problem, as validated in our experiments.

The LSTM branch is designed to strengthen the memory ability of RSD4 algorithm, since it allows the agent to incorporate a large amount of network measurement history into its state space to capture the long-term temporal dependencies of actual system dynamics. The LSTM layer is embedded in the second layer of the multilayer perceptron feature extractor. Consequently, our RSD4 algorithm can well handle various partially observable settings. The fully connected branch is designed to capture more information and improve expressiveness in the current time slot, which provides subsequent layers with more direct access to the current state without requiring information to filter through the LSTM branch. This makes RSD4 more sensitive to the current system state, so that abrupt changes of environmental conditions, e.g., a sudden burst of arrivals or temporal channel condition degradation, can be detected rapidly (shown in Section VI-C.3). Overall, the combination of fully connected and LSTM branches in the architecture of RSD4 provides a good balance between capturing long-term dependencies and reacting to current system states, which are critical for achieving good performance in real-world scheduling problems.

Remark 6: The architecture depicted in Figure 6 is designed to address the challenge of partial observability, which is a common issue in many real-world scenarios. Traditional non-DRL methods and even some non-recurrent

TABLE I
SUMMARY OF ARRIVALS AND CHANNEL STATES

User	Rate	τ_i	Character	Avg. Channel
1	1.96	6	File Transmission	1.79
2	0.91	6	Online Forum	1.83
3	2.46	1	VR Gaming	1.82
4	0.70	1	Text Communication	1.77

DRL algorithms are not explicitly equipped to handle such situations. However, our proposed POMDP formulation and the RSD4 *algorithm leverage the memory mechanism enabled by the LSTM layer, allowing the agent to learn from a batch of historical observations and better capture the long-term temporal dependencies of the underlying dynamics. This enables the agent to effectively handle potentially non-stationary and partially observable environments, while reducing the impact of temporal variability.*

VI. EXPERIMENT RESULTS

We extensively tested RSD4 on both real-world and simulated datasets, averaging results over five runs. We set up the environment in Section VI-A, comparing RSD4 with various algorithms in Section VI-B. We also evaluated the scalability of RSD4 in Section VI-D under scenarios with large user scales and multihop networks, by utilizing the user-level decomposition and node-level merging. An ablation study of the RSD4 design is given in Section VI-E.

A. Environment Setup

This subsection specifies the setup of the single-hop network illustrated in Figure 2 and the multihop environment is constructed in a similar way. In our setup, one interaction step equals one-time slot in our simulated environment. We recommend setting episode length $T = 100$ and episode number $M = 300$ to balance performance and computation complexity, with detailed ablation study in Appendix C.

Arrivals: The arrivals are given by an LTE dataset [34], which records the traffic flow of mobile carriers' 4G LTE network for approximately one year. We construct an environment with four types of arrivals given by the LTE dataset, which are visualized in Figure 7a. The characteristics of the selected data records are provided in Table I, which simulates four representative tasks, i.e., file transmission, online forum, VR gaming, and text communication, according to their rates and delay requirements.

Service Channel Conditions: The channel states used in the experiments are obtained from a wireless 2.4GHz dataset [35]. This dataset records the received signal strength indicator (RSSI) in the check-in hall of Shenzhen Baoan International Airport. Each RSSI value is quantized into four levels, resulting in four possible channel states. These states are visualized in Figure 7b, and their average values are provided in Table I. The assumption made in the experiments is that the service channel conditions are unknown, thereby simulating a partially observable scheduler.

Service Outcomes: We adopt the same assumption for the probability of successful service as presented in [13]. Specifically, the successful service probability for user i under channel state c and allocated resource e is modeled as follows:

$$P_i(e,c) = \frac{2}{1+\exp[-2e/(l_i^3 c)]} - 1, \tag{17}$$

where l_i represents the distance between user i and the server. This modeling of service probability corresponds to a wireless downlink system, where e can be interpreted as the transmission power of the antenna for transmitting the current packet. Furthermore, the resource constraint E_0 signifies that the average power consumption of the antenna stays within the limit of E_0. While we present the form of service probability in Eq. (17), it is worth noting that alternative assumptions for successful service probability can also be considered. Unlike classical optimization-based methods [10], [11], RSD4 does not require linearity or convexity of service probability, since RSD4 is a data-driven approach that employs neural networks to automatically learn the underlying system dynamics.

B. Performance Comparison

We compare the performance of our RSD4 algorithm with existing DRL and classical non-DRL methods in this subsection. The benchmark DRL algorithms used in our experiments are Twin Delayed DDPG (TD3) [32] and Softmax DDPG (SD3) [29], both of which are state-of-the-art DRL algorithms. We apply them with Lagrangian dual to ensure average resource constraints, similar to our RSD4 algorithm in Section III-B. The non-DRL algorithms used in our experiments include Programming, Uniform, and Earliest Deadline First (EDF) [36], which are described below:

(i) Programming: We solve the following static constrained programming problem $\mathcal{P}s$ for each time slot t with the resource constraint E_0 using convex programming [37]:

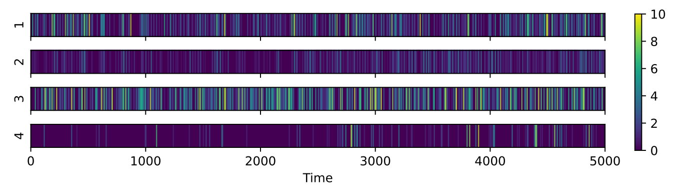
$$\begin{aligned}\mathcal{P}_s : \max_{\boldsymbol{e}} \ & \sum_{i=1}^{N} \beta_i \sum_{\tau=1}^{\tau_i} B_\tau^i(t) P_i(e_i^\tau(t), c_i(t)) \\ \text{s.t.} \ & \sum_{i=1}^{N} \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t) \le E_0\end{aligned} \tag{18}$$

Since the programming method handles the average resource limit E_0 by setting it as a hard resource limit, the optimal value $\mathcal{T}^*_{static}$ for $\mathcal{P}_s$ serves as the static optimal throughput for the Problem $\mathcal{P}$ in Eq. (1). It is important to note that the programming method is infeasible in our experiment environment since the service channel conditions and the successful service probability are unknown beforehand. However, we include this method to highlight the near-optimality of RSD4.

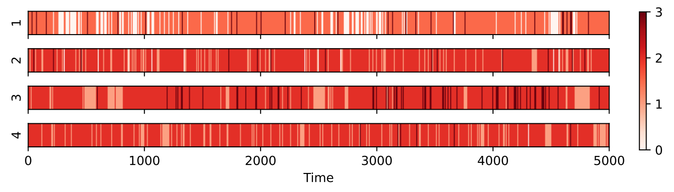
(ii) Uniform: Assign available resources to different packets in the buffer uniformly.

(iii) Earliest Deadline First (EDF) [36]: Assign all resources to packets with the shortest deadline in each queue equally.

To evaluate the case when the resource expenditure at each time may be strictly bounded, we consider the hard resource constraint E_h for each time slot, i.e., scheduling decisions in each time slot cannot exceed E_h. We introduce the reward in Eq.(12) to ensure this, which is motivated by our auxiliary Lagrange function proposed in SectionIII-C, and we set the hard resource constraint $E_h = 2E_0$.

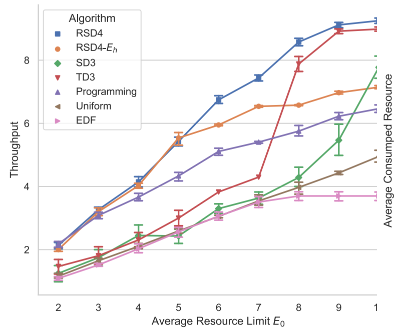


(a) Visulization of four arrivals in 5000 time slots.

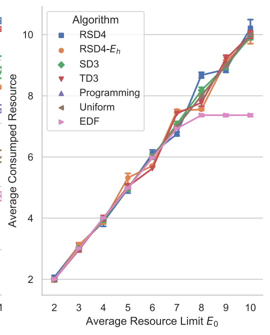


(b) Visulization of channel conditions in 5000 time slots.

Fig. 7. Heat maps of arrivals and channel conditions.



(a) Throughput



(b) Resource Consumption

Fig. 8. Comparison of different algorithms.

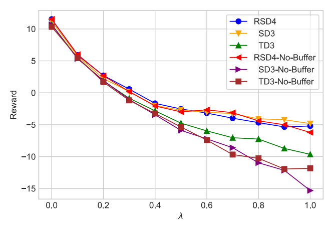
Figure 8 presents a comparison of the performance of the RSD4 algorithm with other scheduling algorithms for a system with four users. The simulations are based on real datasets summarized in Table I, and assume that channels are unobservable to the scheduler. The scenario simulates a multi-user resource-constrained wireless base station that provides delay-constrained scheduling services. Note that since the number of users is small, we do not need to utilize the user-level decomposition mentioned in Section for our RSD4 algorithm. The results in Figure 8 show that the throughput of RSD4-E_h remains relatively close to that of RSD4 for $0 \leq E_0 \leq 5$, but is less than that of RSD4 for $E_0 > 5$. This is because when the average resource limit E_0 is small, the consumed resource in each time slot is almost under the hard resource limit $E_h = 2E_0$. However, when more resources become available, the hard resource limit $E_h = 2E_0$ takes effect, resulting in throughput degradation compared to the case without the hard limit E_h. Furthermore, from Figure 8, it can be observed that RSD4 outperforms all benchmarks in each resource limit. In the small-medium resource regime with $0 \leq E_0 \leq 7$, the static optimality obtained by Programming ranks only lower than RSD4. However, in the large resource regime with $8 \leq E_0 \leq 10$, RSD4 and TD3 outperform classical methods significantly. Moreover, Figure 8 shows that all DRL algorithms satisfy the average resource limit, while earliest deadline first (EDF) fails to fully utilize available resources since packets with the shortest deadline are too few to consume all available resources. The results in Figure 8 demonstrate the superiority of RSD4 over others.

C. Partially Observable Systems

We compared the performance of RSD4 with other DRL algorithms to further validate the effectiveness of its recurrent module. Specifically, we focused on maximizing the Lagrangian function in Eq. (2) using the same environment as in Section VI-B, which is essentially equivalent to the task of throughput maximization. In fact, each average resource limit E_0 corresponds to an optimal λ^*, as explained in Section III-B. Thus, maximizing the Lagrangian function for a given λ is equivalent to throughput maximization under the average resource limit of $E_{\pi^*(\lambda)}$ determined by λ. The standard RSD4 algorithm, presented in Algorithm 1, iteratively employs gradient descent to find the optimal multiplier λ^* for a fixed average resource limit E_0. However, by maximizing the Lagrangian function in Eq. (2) under different values of λ, we can explore a broader solution space and study the system behavior under different constraints parameterized by λ. This study provides a comprehensive understanding about the problem and allows better design and optimization of system performance.

1) Missing Buffer State: We first consider an extreme environment where the buffer state is unobservable by the agent, and only arrivals and service channel information are given. Specifically, the agent's observation $o_t = [\boldsymbol{A}(t), \boldsymbol{c}(t)]$, which implies the agent needs to memorize arrivals and service outcomes across multiple time slots to have accurate estimations of current system states. From Figure 9a, we find that when the buffer state is observable, i.e., the underlying MDP is fully observable, RSD4 and SD3 obtain similar maximum rewards under different λ. However, when the buffer state is missing, RSD4 still achieves almost the same maximum rewards, whereas non-recurrent DRL algorithms SD3 and TD3 suffer from significant performance loss. This result confirms the effectiveness of the recurrent module in capturing temporal dependencies and learning accurate state representations.

2) Unobservable Hidden Factors: Another common type of partial observability comes from unobserved hidden factors in the underlying MDP, e.g., vehicle obstruction will influence in-tunnel wireless propagation channel which is hard to trace [38]. To investigate this type of partial observability, we design an environment where service outcomes are related to the time index, i.e., services are available only when the current time slot t is a multiple of some period, such as wireless communication interfered by periodic jamming signals. In this case, the underlying MDP is partially observable, since the hidden factor, i.e., the period, is unknown to the agent, and the observation is $o_t = [\boldsymbol{B}^1(t), \ldots, \boldsymbol{B}^N(t), \boldsymbol{c}(t)]$. From Figure 9b, we observe that RSD4 outperforms TD3 and SD3 in both tested cases with periods 5 and 10, and the performance gain of RSD4 is more significant in the large λ case, which corresponds to the small resource regime.


(a) Environment w/o buffer state.

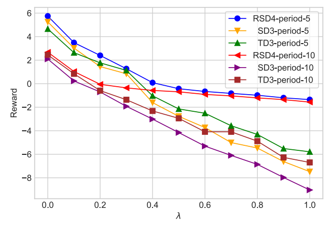
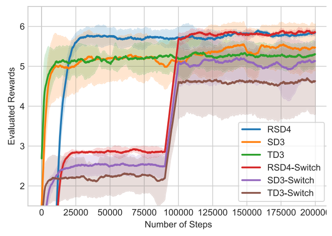

(b) Environment w/ hidden factors.

Fig. 9. Various partially observable settings.


(a) Switch on arrival strength

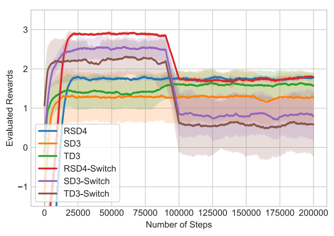

(b) Switch on channel availability

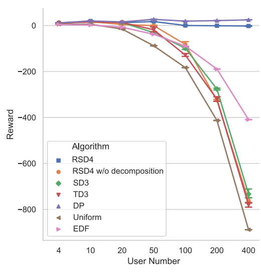
Fig. 10. Experiments on switching environments.

3) Time-Varying Environments: We next investigate partial observability arising from time variability of underlying system dynamics. We introduce two types of time variability by changing environment dynamics during the experiments. Specifically, in one setting, we double the rate of arrivals, and in the other, we change the channel statistics. These two settings simulate sudden changes that can occur in real-world environments. In particular, we compare the performance of different algorithms in maximizing the Lagrangian function under the same fixed λ, to highlight the rapid adaptability of our RSD4 algorithm. In both cases, the channel states are unobservable, and the switch occurs at slot $100,000$. The results are shown in Figure 10, where we compare the evaluated rewards of training processes under the environments with switching in the middle and switching at the very beginning, for the task of maximizing the Lagrangian function under $\lambda = 0.5$. We observe that, after switching, RSD4 quickly reaches the same rewards as those obtained by training in these environments from the very beginning, while TD3 and SD3 both suffer from reward loss after switching, even if they are trained from the beginning for this new environment. This validates the robustness of RSD4 in time-varying environments.

Remark 7: Experiments on these three partially observable environments show the superiority of RSD4 *in addressing partial observability issues. They also demonstrate the benefits of adopting a POMDP formulation in practical scheduling problems, as partial observability can arise from various sources. This comparison also sheds light on understanding the sub-optimality of non-recurrent DRL algorithms or classical non-DRL methods in partially observable environments.*

D. Scalability by Decomposition and Merging

This investigates the scalability of our proposed algorithm by conducting experiments in systems with a large number of users and multihop structures, where the channel states are assumed to be unobservable. Our experimental results


(a) Different user scales.

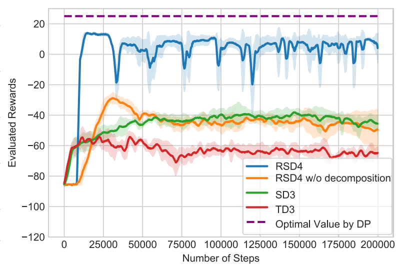

(b) Training process with 50 users.

Fig. 11. Experiments on scalability.

demonstrate that the user-level decomposition and node-level merging techniques in RSD4 make it a highly scalable solution.

1) Large-Scale System: To assess the scalability of our proposed method, we evaluate the performance of RSD4 in comparison with other algorithms on large scale scenarios, i.e. various numbers of users ranging from 4 to 400, as depicted in Figure 11. The channel states are assumed to be unobservable, and we show that the user-level decomposition technique in RSD4 makes it a highly scalable solution. We generated arrivals and channels based on predefined random processes, and all baselines were implemented with the same set of hyperparameters to ensure fairness. To compare the performance of the different algorithms in maximizing the Lagrangian function, we fix the multiplier and compare the rewards of RSD4 with other benchmarks. Consequently, the programming method is not applicable in this case. We generate arrivals and channels based on predefined random processes and obtain the optimal reward using dynamic programming (DP) as presented in [13]. Nevertheless, it is essential to note that computing the optimal throughput by DP is computationally expensive and requires accessing the entire system dynamics, which is impractical in real-world applications. We include this comparison to demonstrate the near-optimality and effectiveness of RSD4.

From Figure 11a, we observe that when the number of users is less than or equal to 10, the performance of various baselines is similar. However, as the number of users increases, all DRL algorithms, including RSD4 without user-level decomposition, as well as Uniform and EDF, fail to achieve optimal performance. In contrast, the decomposed RSD4 consistently outperforms other baselines and achieves near-optimal performance regardless of the number of users. Figure 11b shows the learning curve under 50 users, where only RSD4 achieves near-optimal reward, while RSD4 without state decomposition and other DRL algorithms fail. These results demonstrate the necessity of state decomposition, without which the algorithm's parameter amount grows too much and prohibits efficient training.

We also conducted experiments on a scheduler with 200 users under average resource limits ranging from 1000 to 1600 to exactly compare the throughput obtained by different algorithms in Figure 12. The result shows that RSD4 outperforms most other methods in the resource range.

Remark 8: When the system scale surpasses the hypothesis dimension of the underlying neural network, such as when the number of users exceeds 20 as shown in Figure 11a, the

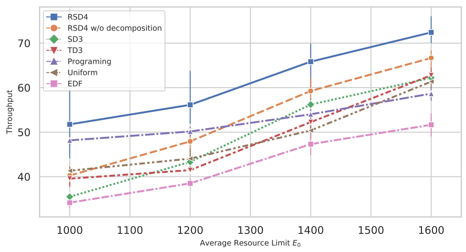
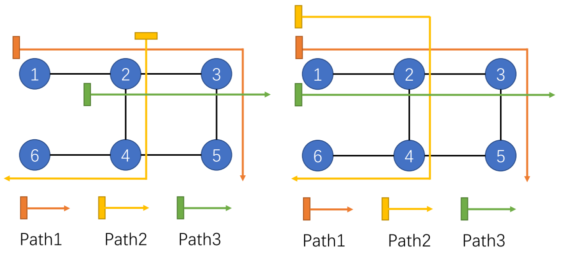
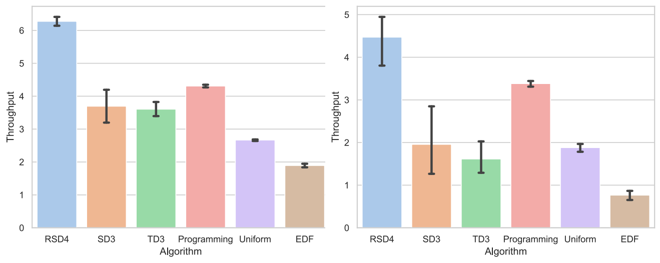


Fig. 12. Performances of different methods with 200 users.



(a) Network 1. (b) Network 2.

Fig. 13. Two different multi-hop network topology.



(a) Results on network 1. (b) Results on network 2.

Fig. 14. Experiments on multihop networks.

performance of DRL algorithms declines rapidly, necessitating neural networks with more hyperparameters. However, as neural networks become more complex, they become more difficult to train and require significantly more computational power. In contrast, with state decomposition, RSD4 *can efficiently control the state dimension while retaining near-optimal performance, showing the effectiveness of user-level decomposition.*

2) *Multihop Network:* Moving on to the multihop network setting, we recall that in Section IV-C we proposed node-level merging as a scheduling method for multihop networks. We now present experiments on two different multihop network topologies depicted in Figures 13a and 13b, respectively. In both networks, $Path_1$, $Path_2$, and $Path_3$ has 3, 4, and 5 flows passing through respectively.

In Figure 13a, there are three paths, $Path_1 = 1 \rightarrow 2 \rightarrow 3 \rightarrow 5$, $Path_2 = 2 \rightarrow 4 \rightarrow 6$, and $Path_3 = 2 \rightarrow 3$. Nodes 1, 2, 3 and 4 have average resource limits 10, 30, 0.3, 3, respectively. In Figure 13b, there are three paths, $Path_1 = 1 \rightarrow 2 \rightarrow 3 \rightarrow 5$, $Path_2 = 1 \rightarrow 2 \rightarrow 4 \rightarrow 6$, and $Path_3 = 1 \rightarrow 2 \rightarrow 3$. Nodes 1, 2, 3 and 4 have average resource limits 10, 30, 0.3, 3, respectively.

The arrival and channel states are drawn from these datasets, and the throughput obtained by different algorithms is shown in Figure 14. We observe that RSD4 achieves the maximum

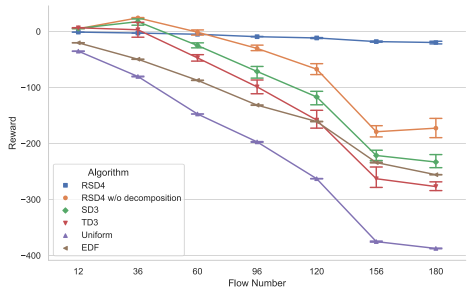


Fig. 15. Rewards on different user scales of multihop networks.

throughput and significantly outperforms other benchmarks, including TD3, SD3, Programming, Uniform, and EDF, which highlights the effectiveness of RSD4 in practical scenarios.

To further validate the scalability of our RSD4 algorithm, we conducted experiments on the multihop network depicted in Figure 13a with a large number of flows. To enable RSD4 to work efficiently in the large-scale setting of multihop networks, we still utilized user-level decomposition with unified training similar to the single-hop case. Specifically, there are $3x$, $4x$, and $5x$ flows passing through $Path_1$, $Path_2$, and $Path_3$, respectively. We tested the algorithm with an increasing number of flows from 12 to 180 (i.e., x ranging from 1 to 15) in Figure 15.[3] The achieved maximum rewards by different algorithms are shown in Figure 15. From the figure, we observe that when the number of flows is less than or equal to 60, the performance of various DRL algorithms is similar, while non-DRL methods perform poorly. However, as the number of flows increases, all DRL algorithms, including RSD4 without decomposition, fail to achieve optimal performance. In contrast, the RSD4 achieves a larger performance gain over other algorithms as the number of flows increases while still saving computational resources by utilizing unified training.

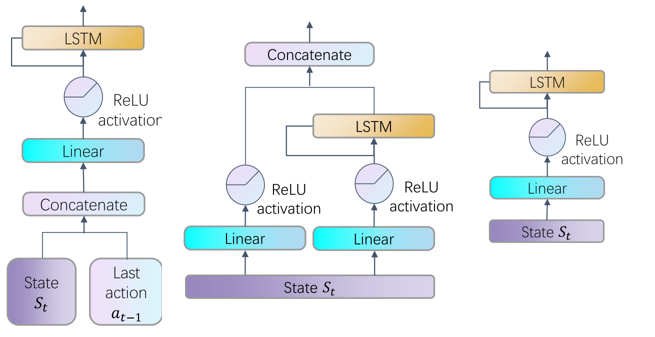
E. Ablation Study

Here we present an ablation study on network architectures, as well as important hyperparameters such as policy update frequency and learning rate. The results of this study will guide us in determining appropriate algorithm parameters. To ensure that the underlying Markov decision process (MDP) is well-defined with time-invariant distributions, we use 20 pairs of simulated arrivals and channels in the environment by pre-specifying their distributions.

We compare different network architectures in Figure 16 with our proposed architecture shown in Figure 6.[4] These architectures are evaluated using RSD4 on the task of maximizing the Lagrangian function in Eq. (2) with $\lambda = 0, 0.1, \ldots, 1$. We chose this range of λ as it corresponds to the possible optimal multiplier λ^* in our environment setting. To ensure comparability across architectures, we normalize the rewards obtained under different λ values.

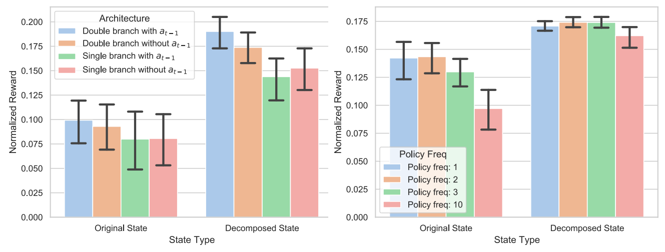
[3]We compare the performance of the different algorithms in maximizing the Lagrangian function here by fixing the multiplier and compare the rewards of RSD4 with other benchmarks.

[4]we only present the first several layers of the critic network, while the last two layers are the same as those shown in Figure 6.



(a) Single-branch w/ a_{t-1} (b) Double-branch w/o a_{t-1} (c) Single-branch w/o a_{t-1}

Fig. 16. Different network architectures (part).

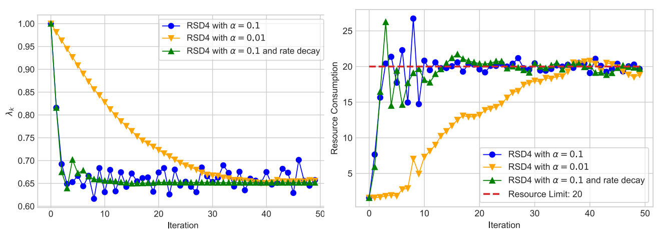


(a) Different network architecture (b) Different policy frequencies

Fig. 17. Experiments of the ablation study.

1) *Network Architectures:* The results are presented in Figure 17a, showcasing the rewards attained using RSD4 with and without the utilization of user-level decomposition. Our analysis demonstrates that the double-branch architecture achieved maximum rewards in both scenarios. Furthermore, we discovered that incorporating the previous action a_{t-1} as an input to the LSTM led to further enhanced rewards, as illustrated in Figure 16. This emphasizes the advantage of our proposed double-branch architecture and suggests that integrating the previous action can better capture underlying correlations over time. Moreover, for both the original system state and the decomposed system state, we employed two neural networks with identical hidden dimensions. Remarkably, when equipped with user-level decomposition, the algorithm secures more rewards even with only 20 users, underscoring the potential stability facilitated by user-level decomposition. This observation aligns with the trend shown in Figure 11a, where RSD4 without user-level decomposition trailed behind RSD4 when the number of users exceeds 20.

2) *Policy Delay:* To prevent training divergence caused by overestimating a poor policy [32], RSD4 adopts a delayed policy update mechanism, as shown on line 25 in Algorithm 1. In this mechanism, the policy network is updated at a lower frequency than the value network, allowing for a reduction in error before introducing a policy update. We evaluate different policy frequencies on the same simulated environment as described in Section VI-E.1, and the results are presented in Figure 17b. Our experiment shows that a moderate policy frequency of 2 improves the peak reward in both the original and decomposed cases. Similar to Figure 17a, when equipped with user-level decomposition, the algorithm achieves higher rewards with 20 users.



(a) λ_k (b) Resouce consumption

Fig. 18. Iterations under different learning rates.

3) *Learning Rate:* The choice of learning rate α in Algorithm 1 is critical. We conduct experiments using different values of α, as shown in Figure 18, which displays the iteration of λ_k and resource consumption, respectively. Our analysis indicates that a learning rate of 0.1 is too large, leading to large fluctuations in λ_k. On the other hand, a small learning rate of 0.01 eliminates these fluctuations but converges too slowly. To address this issue, we adopt a decaying learning rate strategy that halves the learning rate when λ flips in three consecutive time slots, i.e., $\alpha_{k+1} = \alpha_k$ if $\lambda_{k+1} \geq \lambda_k \geq \lambda_{k-1}$ or $\lambda_{k+1} \leq \lambda_k \leq \lambda_{k-1}$; otherwise, $\alpha_{k+1} = \alpha_k/2$.

VII. Conclusion

This paper presents a novel approach to the problem of multi-user latency-constrained scheduling with average resource constraints. To overcome the challenges of partial observability and scalability, we propose a data-driven method based on a POMDP formulation, called RSD4, which ensures resource and delay constraints by leveraging Lagrangian dual and delay-sensitive queues, respectively. RSD4 addresses the partially observable issue through the use of a recurrent network module. It also enables robust value estimation with the softmax operator and introduces user-level decomposition and node-level merging techniques to ensure scalability. Our experimental evaluations, conducted on both simulated environments and real-world datasets, demonstrate that RSD4 significantly outperforms existing DRL/non-DRL-based benchmarks. Furthermore, our results show that RSD4 is highly-scalable and robust to various system dynamics and partially observable settings.

Appendix A
Proof of Lemma 1

Before presenting Lemma 1, we first define a truncation operation in Definition 9.

Definition 9 (Truncation Operation): Let $a_t = [\boldsymbol{e}_1(t), \boldsymbol{e}_2(t), \dots, \boldsymbol{e}_N(t)]$ *be an output of a policy* π, *and* $E_h > 0$ *be a hard limit. The truncated action* $\overline{a}_t$ *with respect to* E_h *is defined as:*

$$\overline{a}_t = \begin{cases} a_t, & \text{if } \sum_{i=1}^{N} \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t) \leq E_h \\ \boldsymbol{0}, & \textit{Otherwise.} \end{cases}$$

Now we are ready to present Lemma 1. *Proof:* First, we prove $\overline{g}(\lambda) \leq \widetilde{g}(\lambda)$ for any $\lambda > 0$. For any policy π whose outputs satisfy the hard resource limit $\sum_{i=1}^N \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t) \le E_h$, we have

$$\sum_{i=1}^N \left[\beta_i d_i(t) - \lambda \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i^\top(t)\right] = \mathbb{1}\left\{\sum_{i=1}^N \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t) \le E_h\right\}\sum_{i=1}^N[\beta_i d_i(t) - \lambda \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i^\top(t)] \tag{19}$$

for any $1 \le t \le T$. Thus, we have

$$\begin{aligned}\overline{g}(\lambda) &= \max_{\pi:\sum_{i=1}^N \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)\le E_h,\forall 1\le t\le T} \mathcal{L}(\pi,\lambda)\\ &= \max_{\pi:\sum_{i=1}^N \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)\le E_h,\forall 1\le t\le T} \widetilde{\mathcal{L}}(\pi,\lambda)\\ &\le \max_\pi \widetilde{\mathcal{L}}(\pi,\lambda) = \widetilde{g}(\lambda),\end{aligned} \tag{20}$$

where the first equality holds by Eq. (5), the second equality holds by Eq. (19), the first inequality holds by definition, and the last equality holds by Eq. (7).

Secondly, we prove $\widetilde{g}(\lambda) \le \overline{g}(\lambda)$ for any $\lambda > 0$. Denote the actions of $\overline{\pi}^*(\lambda)$ and $\widetilde{\pi}^*(\lambda)$ at time t as $\{\overline{\boldsymbol{e}}_i(t)\}_{i\le 1\le N}$ and $\{\widetilde{\boldsymbol{e}}_i(t)\}_{i\le 1\le N}$, respectively. We have

$$\begin{aligned}&\mathbb{1}\left\{\sum_{i=1}^N \widetilde{\boldsymbol{e}}_i^\top(t)\boldsymbol{B}_i(t)\le E_h\right\}\sum_{i=1}^N\left[\beta_i d_i(t) - \lambda\widetilde{\boldsymbol{e}}_i^\top(t)\boldsymbol{B}_i^\top(t)\right]\\ &= \mathbb{1}\left\{\sum_{i=1}^N \overline{\boldsymbol{e}}_i^\top(t)\boldsymbol{B}_i(t)\le E_h\right\}\sum_{i=1}^N\left[\beta_i d_i(t) - \lambda\overline{\boldsymbol{e}}_i^\top(t)\boldsymbol{B}_i^\top(t)\right]\\ &= \sum_{i=1}^N\left[\beta_i d_i(t) - \lambda\overline{\boldsymbol{e}}_i^\top(t)\boldsymbol{B}_i^\top(t)\right]\end{aligned} \tag{21}$$

for any $1 \le t \le T$, where the first equality holds by considering the following two cases: Case i): If $\sum_{i=1}^N \widetilde{\boldsymbol{e}}_i^\top(t)\boldsymbol{B}_i(t) > E_h$, then $\mathbb{1}\{\sum_{i=1}^N \widetilde{\boldsymbol{e}}_i^\top(t)\boldsymbol{B}_i(t) \le E_h\} = 0$. In this case $\overline{\boldsymbol{e}}_i(t) = \boldsymbol{0}$ for any $1 \le i \le N$ and all $d_i(t) = 0$. Thus, both hands of the first equality in Eq. (21) are zero. Case ii): Otherwise, $\widetilde{\boldsymbol{e}}_i(t) = \overline{\boldsymbol{e}}_i(t)$ for any $1 \le i \le N$ and the equality holds. Besides, the second equality in Eq. (21) holds since $\overline{\pi}^*(\lambda)$ is obtained by the truncation operation such that $\{\overline{\boldsymbol{e}}_i(t)\}_{i\le 1\le N}$ always satisfies the hard resource limit. Thus,

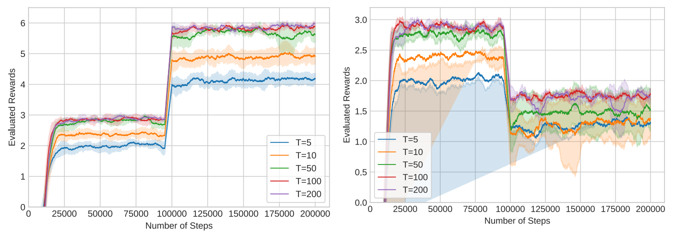
$$\begin{aligned}\widetilde{g}(\lambda) &= \widetilde{\mathcal{L}}(\widetilde{\pi}^*(\lambda),\lambda) = \mathcal{L}(\overline{\pi}^*(\lambda),\lambda)\\ &\le \max_{\pi:\sum_{i=1}^N \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i(t)\le E_h,\forall 1\le t\le T}\mathcal{L}(\pi,\lambda) = \overline{g}(\lambda),\end{aligned} \tag{22}$$

where the first equality holds by Eq. (7), the second equality holds by Eq. (21), the inequality holds by definitions, and the last equality holds by Eq. (5). Combining Eq. (20) and Eq. (22) gives $\overline{g}(\lambda) = \widetilde{g}(\lambda)$ for any $\lambda > 0$, and the truncated policy $\overline{\pi}^*(\lambda)$ is the maximizer for $\overline{g}(\lambda)$. □

APPENDIX B
PROOF OF LEMMA 4

Proof: For any user i and any λ_i, we have

$$\mathcal{L}_i(\pi_i^*(\lambda_i),\lambda_i) \ge \max_{\pi_i}\mathcal{L}_i(\pi_i,\lambda_i), \tag{23}$$



(a) Switch on arrival strength (b) Switch on channel availability

Fig. 19. Ablation of episode length T on switching environments.

by the definition of $\pi_i^*(\lambda_i)$. Summing Eq. (23) over i from 1 to N for a fixed λ yields

$$\begin{aligned}&\sum_{i=1}^N \mathcal{L}_i(\pi_i^*(\lambda_i),\lambda_i) \ge \sum_{i=1}^N \max_{\pi_i}\mathcal{L}_i(\pi_i,\lambda_i)\\ &= \sum_{i=1}^N \max_{\pi_i}\lim_{T\to\infty}\frac{1}{T}\left[\sum_{t=1}^T \beta_i d_i(t) - \lambda_i \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i^\top(t)\right]\\ &\ge \max_\pi \sum_{i=1}^N \lim_{T\to\infty}\frac{1}{T}\left[\sum_{t=1}^T \beta_i d_i(t) - \lambda_i \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i^\top(t)\right]\\ &= \max_\pi \lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^T\sum_{i=1}^N\left[\beta_i d_i(t) - \lambda \boldsymbol{e}_i^\top(t)\boldsymbol{B}_i^\top(t)\right]\\ &= \max_\pi\left[\mathcal{L}(\pi,\lambda) - \lambda E_0\right],\end{aligned} \tag{24}$$

where the first equality holds by Eq. (13), the second inequality holds by definitions of π and π_i, the second equality holds by interchanging summation and limitation, and last equality holds by Eq. (2). Let $\lambda_i = \lambda$ for any i. We have

$$\begin{aligned}\mathcal{L}(\widehat{\pi}(\lambda),\lambda) - \lambda E_0 &= \sum_{i=1}^N \mathcal{L}_i(\pi_i^*(\lambda_i),\lambda_i)\\ &\ge \max_\pi\left[\mathcal{L}(\pi,\lambda) - \lambda E_0\right],\end{aligned} \tag{25}$$

where the inequality holds by Eq. (2), Eq. (13), and the definition of the aggregated policy $\widehat{\pi}(\lambda)$, and the inequality holds by Eq. (24). Since λ and E_0 are both constants, we have $\mathcal{L}(\widehat{\pi}(\lambda),\lambda) = \max_\pi \mathcal{L}(\pi,\lambda)$, implying that $\widehat{\pi}(\lambda)$ is the optimal policy that maximizes $\mathcal{L}(\pi,\lambda)$. □

APPENDIX C
ABLATION STUDY ON EPISODE LENGTH

We provide an ablation study on the episode length T of two time-varying environments in Section VI-C.3 and the results are given in Figure 19. As the agent operates episodically, episode length T represents the time slot range sequentially input to the network via the LSTM module. From Figure 19, we observe that when $T \le 100$, RSD4's performance improves with an increasing T. However, beyond $T = 100$, the performance reaches a plateau, indicating that $T = 100$ provides a sufficient range for the agent to extract cross-time slot information. Since a larger T induces larger computation complexity in the LSTM module [33] for each decision-making step, we recommend utilizing an episode length of $T = 100$ to balance performance and computational complexity effectively.

In addition, as illustrated in Figure 19, approximately $30,000$ total time steps are adequate for RSD4's convergence in different episode lengths. Thus, we take $T = 100$ and $M = 300$ in our experiments in Section VI (so that the total training step is $M \cdot T = 30,000$). This recommendation of $T = 100$ and $M = 300$ aims to strike a balance between convergence and computational efficiency.

References

[1] S. Ma, "Fast or free shipping options in online and omni-channel retail? The mediating role of uncertainty on satisfaction and purchase intentions," *Int. J. Logistics Manag.*, vol. 28, no. 4, pp. 1099–1122, Nov. 2017.

[2] H. Zhang, W. Li, S. Gao, X. Wang, and B. Ye, "ReLeS: A neural adaptive multipath scheduler based on deep reinforcement learning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 1648–1656.

[3] R. Bhattacharyya et al., "QFlow: A reinforcement learning approach to high QoE video streaming over wireless networks," in *Proc. 20th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, Jul. 2019, pp. 251–260.

[4] L. Qu, C. Assi, K. Shaban, and M. J. Khabbaz, "A reliability-aware network service chain provisioning with delay guarantees in NFV-enabled enterprise datacenter networks," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 554–568, Sep. 2017.

[5] N.-N. Dao, A.-T. Tran, N. H. Tu, T. T. Thanh, V. N. Q. Bao, and S. Cho, "A contemporary survey on live video streaming from a computation-driven perspective," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 1–38, Jan. 2022.

[6] R. Xie, Q. Tang, C. Liang, F. R. Yu, and T. Huang, "Dynamic computation offloading in IoT fog systems with imperfect channel-state information: A POMDP approach," *IEEE Internet Things J.*, vol. 8, no. 1, pp. 345–356, Jan. 2021.

[7] C.-P. Li and M. J. Neely, "Network utility maximization over partially observable Markovian channels," *Perform. Eval.*, vol. 70, nos. 7–8, pp. 528–548, Jul. 2013.

[8] L. Huang, S. Zhang, M. Chen, and X. Liu, "When backpressure meets predictive scheduling," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2237–2250, Aug. 2016.

[9] Z. Scully, M. Harchol-Balter, and A. Scheller-Wolf, "Simple near-optimal scheduling for the M/G/1," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 48, no. 1, pp. 37–38, Jul. 2020.

[10] I.-H. Hou and P. Kumar, "Utility-optimal scheduling in time-varying wireless networks with delay constraints," in *Proc. 11th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2010, pp. 31–40.

[11] C. Li, P. Hu, Y. Yao, B. Xia, and Z. Chen, "Optimal multi-user scheduling for the unbalanced full-duplex buffer-aided relay systems," *IEEE Trans. Wireless Commun.*, vol. 18, no. 6, pp. 3208–3221, Jun. 2019.

[12] R. Singh and P. R. Kumar, "Throughput optimal decentralized scheduling of multihop networks with end-to-end deadline constraints: Unreliable links," *IEEE Trans. Autom. Control*, vol. 64, no. 1, pp. 127–142, Jan. 2019.

[13] K. Chen and L. Huang, "Timely-throughput optimal scheduling with prediction," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2457–2470, Dec. 2018.

[14] S. Pan and Y. Chen, "Energy-optimal scheduling of mobile cloud computing based on a modified Lyapunov optimization method," *IEEE Trans. Green Commun. Netw.*, vol. 3, no. 1, pp. 227–235, Mar. 2019.

[15] L. Lv et al., "Contract and Lyapunov optimization-based load scheduling and energy management for UAV charging stations," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 3, pp. 1381–1394, Sep. 2021.

[16] X. Fu et al., "Conmap: A novel framework for optimizing multicast energy in delay-constrained mobile wireless networks," in *Proc. 18th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2017, pp. 1–10.

[17] K. Koosheshi and S. Ebadi, "Optimization energy consumption with multiple mobile sinks using fuzzy logic in wireless sensor networks," *Wireless Netw.*, vol. 25, pp. 1215–1234, Apr. 2019.

[18] V.-T. Pham, T. N. Nguyen, B.-H. Liu, M. T. Thai, B. Dumba, and T. Lin, "Minimizing latency for data aggregation in wireless sensor networks: An algorithm approach," *ACM Trans. Sensor Netw.*, vol. 18, no. 3, pp. 1–21, Aug. 2022.

[19] H. Xue, H. Chen, Q. Dai, K. Lin, J. Li, and Z. Li, "CSCT: Charging scheduling for maximizing coverage of targets in WRSNs," *IEEE Trans. Computat. Social Syst.*, early access, May 6, 2022, doi: 10.1109/TCSS.2022.3169780.

[20] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 197–210.

[21] Y. S. Nasir and D. Guo, "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2239–2250, Oct. 2019.

[22] F. Meng, P. Chen, L. Wu, and J. Cheng, "Power allocation in multi-user cellular networks: Deep reinforcement learning approaches," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6255–6267, Oct. 2020.

[23] H. Song, M. Xiao, J. Xiao, Y. Liang, and Z. Yang, "A POMDP approach for scheduling the usage of airborne electronic countermeasures in air operations," *Aerosp. Sci. Technol.*, vol. 48, pp. 86–93, Jan. 2016.

[24] A. Gong, T. Zhang, H. Chen, and Y. Zhang, "Age-of-information-based scheduling in multiuser uplinks with stochastic arrivals: A POMDP approach," 2020, *arXiv:2005.05443*.

[25] D. Bertsekas, A. Nedic, and A. Ozdaglar, *Convex Analysis and Optimization*, vol. 1. Nashua, NH, USA: Athena Scientific, 2003.

[26] C. Xu, S. Liu, C. Zhang, Y. Huang, and L. Yang, "Joint user scheduling and beam selection in mmWave networks based on multi-agent reinforcement learning," in *Proc. IEEE 11th Sensor Array Multichannel Signal Process. Workshop (SAM)*, Jun. 2020, pp. 1–5.

[27] F. Zhou, L. Feng, M. Kadoch, P. Yu, W. Li, and Z. Wang, "Multiagent RL aided task offloading and resource management in Wi-Fi 6 and 5G coexisting industrial wireless environment," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 2923–2933, May 2022.

[28] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," 2015, *arXiv:1512.04455*.

[29] L. Pan, Q. Cai, and L. Huang, "Softmax deep double deterministic policy gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 11767–11777.

[30] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[31] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real transfer of robotic control with dynamics randomization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 3803–3810.

[32] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. ICML*, 2018, pp. 1587–1596.

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[34] N. Loi. (2018). *Predict Traffic of Lte Network | Kaggle*. Accessed: Jul. 2021. [Online]. Available: https://www.kaggle.com/naebolo/predict-traffic-of-lte-network

[35] W. Taotao, X. Jiantao, X. Wensen, C. Yucheng, and Z. Shengli. (2021). *Wireless Signal Strength on 2.4 GHZ (WSS24) Dataset*. Accessed: Feb. 17, 2022. [Online]. Available: https://github.com/postman511/Wireless-Signal-Strength-on-2.4GHz-WSS24-dataset

[36] K. M. F. Elsayed and A. K. F. Khattab, "Channel-aware earliest deadline due fair scheduling for wireless multimedia networks," *Wireless Pers. Commun.*, vol. 38, no. 2, pp. 233–252, Jul. 2006.

[37] K. Shen and W. Yu, "Fractional programming for communication systems—Part I: Power control and beamforming," *IEEE Trans. Signal Process.*, vol. 66, no. 10, pp. 2616–2630, May 2018.

[38] J. Song, W. Wang, and R. Ibrahim, "The impact of obstruction by vehicle on in-tunnel wireless propagation channel," in *Proc. IEEE 4th Int. Conf. Electron. Inf. Commun. Technol. (ICEICT)*, Aug. 2021, pp. 572–576.

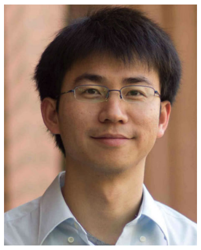
Pihe Hu received the B.E. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University (SJTU), Shanghai, China, in 2019. He is currently pursuing the Ph.D. degree with the Institute for Interdisciplinary Information Sciences (IIIS), Beijing, China, under the supervision of Prof. Longbo Huang. His research interests include reinforcement learning, sparse neural networks, and stochastic optimization.

Yu Chen received the B.S. degree from the Department of Mathematical Science, Tsinghua University, Beijing, China, in 2023. He is currently pursuing the Ph.D. degree with the Institute for Interdisciplinary Information Sciences (IIIS), Beijing, under the supervision of Prof. Longbo Huang. His research interests include the theory and application of reinforcement learning.

Ling Pan received the Ph.D. degree from the Institute for Interdisciplinary Information Sciences, Tsinghua University, in 2022, under the supervision of Prof. Longbo Huang. She also spent time at Stanford University, the University of Oxford, and Microsoft Research Asia, during the Ph.D. study. She was a Post-Doctoral Fellow with MILA under the supervision of Prof. Yoshua Bengio. She is currently an Assistant Professor with the Department of Electronic and Computer Engineering and the Department of Computer Science and Engineering (by courtesy), The Hong Kong University of Science and Technology. Her research interests include generative flow networks (GFlowNets), deep reinforcement learning, and multi-agent systems.

Zhixuan Fang (Member, IEEE) received the B.S. degree in physics from Peking University, China, in 2013, and the Ph.D. degree in computer science from Tsinghua University, Beijing, China, in 2018. He is currently a tenure-track Assistant Professor with the Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University. His research interests include the design and analysis of multi-agent systems, blockchain, and networked systems.

Fu Xiao (Member, IEEE) received the Ph.D. degree in computer science and technology from the Nanjing University of Science and Technology, Nanjing, China, in 2007. He is currently a Professor and the Ph.D. Supervisor with the School of Computers, Nanjing University of Posts and Telecommunications. His research interests include wireless sensor networks and mobile computing. He is a member of the Association for Computing Machinery.



Longbo Huang (Senior Member, IEEE) is currently a Professor with the Institute for Interdisciplinary Information Sciences (IIIS), Tsinghua University, Beijing, China. He has held visiting positions at the LIDS Laboratory, MIT, CUHK, Bell-Labs France, and Microsoft Research Asia (MSRA). He was a Visiting Scientist at the Simons Institute for the Theory of Computing in Fall 2016. He is an ACM Distinguished Scientist, a CCF Distinguished Member, an IEEE ComSoc Distinguished Lecturer, and an ACM Distinguished Speaker. He received the Outstanding Teaching Award from Tsinghua University in 2014 and the Google Research Award and the Microsoft Research Asia Collaborative Research Award in 2014. He was selected for the MSRA StarTrack Program in 2015. He received the ACM SIGMETRICS Rising Star Research Award in 2018. He serves/served on the editorial board for IEEE Journal on Selected Areas in Communications, IEEE Transactions on Communications, ACM Transactions on Modeling and Performance Evaluation of Computing Systems, IEEE/ACM Transactions on Networking, *Performance Evaluation* (Elsevier), and IEEE Transactions on Pattern Analysis and Machine Intelligence.