Performance Evaluation 152 (2021) 102237

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

Online learning for hierarchical scheduling to support network slicing in cellular networks[☆]

Jianhan Song [*], Gustavo de Veciana, Sanjay Shakkottai

ECE Department, The University of Texas at Austin, Austin, TX 78712, USA

ARTICLE INFO

Article history:
Available online 5 October 2021

Keywords:
Wireless networks
Network slicing
Online learning
Bandit algorithms

ABSTRACT

We study a learning-based hierarchical scheduling framework in support of network slicing for cellular networks. This addresses settings where users and/or service classes are grouped into slices, and resources are allocated hierarchically. The hierarchy is implemented by combining a slice-level scheduler which allocates resources to slices, and flow-level schedulers within slices which opportunistically allocate resources to users/services. Optimizing the slice-level scheduler to maximize system utility is typically hard due to underlying heterogeneity and uncertainty in user channels and performance requirements. We address this by reformulating the problem as an online black-box optimization where slice-level schedulers (parameterized by a weight vector) combined with flow-level schedulers result in user/service level stochastic rewards representing performance fitness; the goal is to learn the best weight vector. We develop a bandit algorithm based on queueing cycles by building on Hierarchical Optimistic Optimization (HOO). The algorithm guides the system to improve the choice of the weight vector based on observed rewards. Theoretical analysis of our algorithm shows a sub-linear regret with respect to an omniscient genie. Finally through simulations, we show that the algorithm adaptively learns the optimal weight vectors when combined with opportunistic and/or utility-maximizing flow-level schedulers.

1. Introduction

The increasing complexity of cellular wireless networks has led to network slicing as a popular paradigm for resource sharing [1]. Network slicing is a coarse resource allocation mechanism that partitions traffic flows into groups (slices), and allocates network resources (e.g. spectrum) to each of these slices. Network slicing typically operates hand-in-hand with a finer-grain resource manager (flow-level scheduler) that allocates resources among the flows within each slice. Slicing can be used for various reasons including isolating groups from each other in the presence of traffic load fluctuations, or grouping flows with similar Quality of Service (QoS) requirements, so that flow-level schedulers can operate across groups of flows with roughly homogeneous requirements.

For example, a network operator might provision a slice for a Mobile Virtual Network Operator (MVNO), e.g., a cheaper cellular provider or to an autonomous driving service (for map downloads to cars and over-the-air software upgrades). Each slice could then be virtually allocated network resources with some guarantees on availability and/or isolation from

[☆] This work was supported by NSF, USA, grants CNS-1731658, CNS-1718089, and CNS-1910112, Army Futures Command Grant, USA W911NF1920333 and the Wireless Networking and Communications Group Industrial Affiliates Program, USA.

[*] Corresponding author.
E-mail addresses: jianhansong@utexas.edu (J. Song), deveciana@utexas.edu (G. de Veciana), sanjay.shakkottai@utexas.edu (S. Shakkottai).

https://doi.org/10.1016/j.peva.2021.102237

traffic fluctuations from other traffic sharing the network. As another example, a carrier might create a slice to support mobile users requiring real-time video flows for which the desired QoS metric is tied to meeting packet deadlines, and a slice for users carrying out file downloads for which QoS is better tied to mean delays and/or flow throughput. Since the QoS requirements within each slice are similar, such grouping allows simpler flow-level (within each slice) scheduling algorithms.

In this paper, we adopt a hierarchical online learning approach to network slicing that is driven by user-feedback in the form of rewards. Given a collection of slices (each slice defined through a collection of flows, and with QoS and spectrum-share requirements), we develop a slice-level scheduler (top of the hierarchy) that dynamically allocates resources to each slice based on the observed rewards from mobile users within each slice. This slice-level scheduler allocates resources by dynamically selecting weights for each slice, with these weights specifying a share of spectrum for each slice through an allocation mechanism such as a *Generalized Processor Sharing (GPS)* scheduler [2].

Further, within each slice, a flow-level scheduler (such as the MaxWeight rule [3]) allocates channel resources to individual flows. Thus, the reward obtained from a slicing allocation depends both on the sharing of spectrum for each slice and the individual allocations within each slice. By treating the transformation from weight selection to reward accumulation[1] as a *blackbox function*, we build on bandit-based blackbox optimization methods to develop adaptive slicing mechanisms. Our main contributions are as follows:

1. Hierarchical Scheduling through Blackbox Optimization: We consider a hierarchical scheduling framework in which a slice-level scheduler parameterized by a weight vector $\boldsymbol{w}$. For any choice of the weight vector $\boldsymbol{w}$, the system observes a mean reward rate associated with users' performance/utility — this mapping from weights to reward rates is represented as a function $f(\boldsymbol{w})$. Due to complexity and dynamics of such systems, $f(\boldsymbol{w})$ is analytically hard to optimize and the problem can be better studied as a blackbox optimization. Using a multi-armed bandit framework, where the (continuous-valued) weights correspond to the arms of the bandit and the corresponding arm-rewards accrue from (noisy) user feedback, we develop algorithms that explore the choice of weights to adaptively optimize the blackbox function.

2. CHOOC Algorithm and Analysis: We propose *Cycle-Based HOO with Clipping* (CHOOC) algorithm, a modified Hierarchical Optimistic Optimization (HOO) algorithm [4] to determine the best weight vector for our blackbox optimization problem. CHOOC operates at the time-scale of queueing cycles (idle + busy period); the queue dynamics and rewards are conditionally (given the weight parameter) independent over cycles under proper assumptions.[2] A single exploration sample of $f(\cdot)$ corresponds to selecting a weight vector $\boldsymbol{w}$, using these weights to allocate spectrum resources (to slices) via the associated slice-level scheduler, in turn allow predetermined flow-level schedulers to assign resources to individual users, and finally collecting the aggregate reward from active users over a queueing cycle.

From a technical perspective, as compared to HOO we address two additional challenges: *(i) Ratio of Rewards:* Since the length of queueing cycles are random and depend on the action (the weight vector $\boldsymbol{w}$), our reward rate is described through a ratio of two random summations — reward accrued over cycles divided by the cumulative cycle lengths — thus, we need to control the associated uncertainty which does not directly fit the standard HOO model (because ratios of sums differs from the sum of ratios). *(ii) Sub-Exponential Rewards and Unstable Queues:* Unlike the sub-Gaussian reward setting of HOO, queueing cycle lengths are either sub-exponential (if $\boldsymbol{w}$ results in stable queues), or can be infinite if the queues become unstable. Thus, we need to clip cycles (i.e. truncate overly long cycles by dropping packets), but must do so with a negligible rate of clipping (to minimize drops). By properly addressing these issues, our theoretical analysis recovers a sub-linear regret, which is of the same order as HOO.

3. Empirical Evaluation: We simulate our algorithm in various wireless settings, which include different slice partitions and heterogeneous performance metrics of user packets, such as mean delay, deadline requirement, and total throughput (for infinitely backlogged users). The experiments show our algorithm is able to locate the optimal weight after a reasonable amount of exploration, and in particular, demonstrate its potential to solve difficult problems, where simple heuristics are either hard to design or under-perform. The simulation examples exhibit the power of our framework in tackling the optimization of performance tradeoffs via hierarchical scheduling across slices, including scenarios where slices' flow level opportunistic schedulers are designed to optimize the sum utility of infinitely backlogged users share resources with users that use queue-based opportunistic schedulers meet queue stability and/or deadline requirements.

1.1. Related work

Wireless Scheduling and Network Slicing. Multi-user wireless scheduling has been studied for decades with numerous designs developed to meet various needs (see [2] for a survey). Meanwhile, network slicing has received substantial attention recently [1,5], introducing new research topics regarding wireless scheduling [6–8]. In this paper, we consider a hierarchical scheduling model for network slicing, which is aligned most to the *network virtualization substrate (NVS)* architecture proposed in [6].

[1] This transformation from weights to rewards occurs through a multi-step process: The weights define spectrum shared to each slice through the slice-level GPS scheduler. Within each slice, the flow-level scheduler opportunistically (based on the current channel realization) allocates channel resources to individual flows. The mobile nodes, upon receiving the packets from the flows generate rewards (e.g. meeting delay deadlines), which are aggregated at the slice level and then across slices to result in the instantaneous (noisy) reward for a weight selection.

[2] Note that naively applying HOO by *periodically* (regardless of the queueing backlogs) exploring new weights/collecting feedback will not work due to the lack of (conditional) independence of feedback.

Table 1
Notation.

$\mathcal{U}$	set of users in the system.
$\mathcal{U}_j, j \in [s]$	set of users in slice j, so $\mathcal{U} = \cup_{j\in[s]}\mathcal{U}_j$.
$\mathcal{U}^{\mathrm{b}}, \mathcal{U}^{\mathrm{nb}}$	sets of users with/without infinitely-backlogged queues.
$\boldsymbol{Q}[t] = (Q_i[t])_{i\in\mathcal{U}}$	queue vector denoting the queue lengths at the start of slot t.
$\boldsymbol{A}[t] = (A_i[t])_{i\in\mathcal{U}^{\mathrm{nb}}}$	packet arrival vector for time slot t.
$\boldsymbol{\lambda}$	mean arrival rate across non-backlogged queues ($\boldsymbol{\lambda} \in \mathbb{R}^{\lvert\mathcal{U}^{\mathrm{nb}}\rvert}$).
$\boldsymbol{w} = (w_j)_{j\in[s]}$	weight vector with w_j denoting the resource allocation of slice j.
$\mathsf{HS}(\boldsymbol{w})$	hierarchical scheduling policy with parameter choice $\boldsymbol{w}$.
$\mathcal{W}$	set of permissible weight parameters.
$\mathcal{W}(\boldsymbol{\lambda})$	subset of $\mathcal{W}$ in which any weight choice for $\mathsf{HS}(\boldsymbol{w})$ is able to stabilize the system with an arrival rate $\boldsymbol{\lambda}$.
$\mathcal{K}$	long-term capacity region such that any $\boldsymbol{\lambda} \in \mathcal{K}^\circ$ is stabilizable.
$\mathcal{K}_{\mathcal{W}}$	capacity region achieved by the class of schedulers $\{\mathsf{HS}(\boldsymbol{w}) : \boldsymbol{w} \in \mathcal{W}\}$ (we assume $\boldsymbol{\lambda} \in (\mathcal{K}_{\mathcal{W}})^\circ$).
$C^{(\boldsymbol{w})}(n), U^{(\boldsymbol{w})}(n)$	random variables denoting the cycle length and reward for the nth cycle if $\mathsf{HS}(\boldsymbol{w})$ scheduler is selected.
π	an adaptive policy to determine weight and clipping choices.
$\hat{C}^\pi_n, \hat{U}^\pi_n$	observed (possibly truncated) cycle length and reward of nth cycle under policy π.
$f(\cdot)$	reward rate function defined for $\boldsymbol{w} \in \mathcal{W}$.
$\boldsymbol{w}^*$	unique best weight vector, i.e., $\boldsymbol{w}^* := \operatorname{argmax}_{\boldsymbol{w}\in\mathcal{W}} f(\boldsymbol{w})$.
f^*	optimal reward rate, i.e., $f^* := f(\boldsymbol{w}^*)$.
$\mathcal{T}$	tree structure into which $\mathcal{W}$ is partitioned.
(h, i)	node of $\mathcal{T}$ at depth h with index $i \in [2^h]$.
$\mathcal{P}_{h,i}$	subset of $\mathcal{W}$ with which node (h, i) is associated.
$f^*_{h,i}$	supremum of reward rates in node (h, i), i.e, $f^*_{h,i} = \sup_{\boldsymbol{w}\in\mathcal{P}_{h,i}} f(\boldsymbol{w})$.
i^*_h	node index such that (h, i^*_h) contains $\boldsymbol{w}^*$.
$d(\nu, \rho)$	near-optimality dimension with respect to parameters (μ, ρ).

Not surprisingly, attempts at using machine learning (and in particular, reinforcement learning) techniques are made to address the network slicing/scheduling problem due to its complex nature. Some successes have been reported in several application settings, mostly from a practical perspective — see e.g. [9–13] for some recent developments.

Multi-Armed Bandits and Blackbox Optimization. Multi-armed bandits (MAB) are an online learning model with a rich literature on theory and applications [14,15]. In particular, MAB settings have been applied to the problem of *blackbox optimization*, where an algorithm optimizes a function f by sequentially selecting actions (aka inputs to f) and receiving feedback (aka function evaluations). Early works include Zooming [16], HOO [4], DOO [17], StoSOO [18], etc, with recent studies focusing on more refined theoretical guarantees and/or various applications, e.g., [19–22].

Finally, bandit algorithms have also been studied in various wireless problems, such as [23–26]. Furthermore, rewards in the form of ratios appear naturally in queueing settings [27,28]. The authors in [28] developed a UCB-type algorithm that learns the best scheduling policy from amongst a fixed number of policies (finite arms). Our paper uses clipping similar to [28] to truncate cycles to ensure the stability of queues (clipping was originally proposed in [27] for handling heavy tails to improve rewards in budgeted bandits with queueing), but our focus here is on weight-choice over a continuum (infinite arms) for the higher-level slicing problem; thus, our challenges and algorithmic approach are different (a more detailed comparison will be discussed in Section 3).

1.2. Notation

We use bold font characters for vectors and normal font for scalars. Unless stated otherwise, random variables are denoted by capital letters. We denote $\mathbb{1}$ as the $\{0, 1\}$-indicator function and $[n]$ as the set $\{1, 2, \ldots, n\}$. Finally, we use "*i.i.d.*" for "independently and identically distributed" and "*a.s.*" for "almost surely".

2. System model

In this section, we formally present a multi-armed bandit framework to address the parameterized network slicing problem. For convenience, the notation introduced in this section is summarized in Table 1.

2.1. Traffic and service model

We consider a queueing system with a single server (base station) and a set of u users, denoted by $\mathcal{U} = [u]$. The users are further grouped into s slices. For each $j \in [s]$, denote $\mathcal{U}_j$ as the set of user indices associated with slice j.

Each user has an associated packet queue. At any time t, the queue lengths of users in the system are denoted by a random vector $\boldsymbol{Q}[t] = (Q_i[t])_{i\in\mathcal{U}}$, where $Q_i[t]$ is the number of packets of the ith user at the beginning of time slot t. Suppose the system may also include a set of users *with infinitely-backlogged queues*, denoted by $\mathcal{U}^{\mathrm{b}}$. Thus, for any $i \in \mathcal{U}^{\mathrm{b}}$, $Q_i[t] = \infty$ for all t. For users in $\mathcal{U}^{\mathrm{nb}} := \mathcal{U} \setminus \mathcal{U}^{\mathrm{b}}$, we assume $(Q_i[0])_{i\in\mathcal{U}^{\mathrm{nb}}} = \boldsymbol{0}$, and packet arrivals are modeled as a random process $(\boldsymbol{A}[t])_{t\geq 0}$. Here, $\boldsymbol{A}[t] = (A_i[t])_{i\in\mathcal{U}^{\mathrm{nb}}}$ where $A_i[t]$ has a bounded distribution for any $i \in \mathcal{U}^{\mathrm{nb}}$. We assume $(\boldsymbol{A}[t])_{t\geq 0}$ are *i.i.d.* across time and denote the expectation by $\boldsymbol{\lambda}$ ($\boldsymbol{\lambda} \in \mathbb{R}^{|\mathcal{U}^{\mathrm{nb}}|}$).

Wireless channels are time-varying. We model the service rates at t as a random vector $\boldsymbol{S}[t] = (S_i[t])_{i \in \mathcal{U}}$ where $S_i[t]$ denotes the service per plot available to the ith user at t. We assume $(\boldsymbol{S}[t])_{t \geq 0}$ are *i.i.d.* over time and also independent of the queue lengths and the arrival process. Without loss of generality, we assume $\mathcal{U}^{\text{nb}} \neq \phi$ throughout this paper[3]. Let $\mathcal{K}$ ($\mathcal{K} \subset \mathbb{R}^{|\mathcal{U}^{\text{nb}}|}$) denote the *long-term capacity* of the system (see [2]), induced by the distribution of $\boldsymbol{S}[t]$. This suggests for any arrival rate $\boldsymbol{\lambda} \in \mathcal{K}^{\circ}$ (the interior of $\mathcal{K}$), there exists at least one scheduling policy that stabilizes the system (i.e., the average queue lengths are finite).

2.2. Hierarchical scheduler

We consider a hierarchical scheduler operated by the system, which consists of a slice-level scheduler and s flow-level schedulers, one for each slice. The slice-level scheduler determines how many resources (e.g., number of resource blocks) are allocated to each slice for each time slot, which is typically *not* channel-aware (i.e., opportunistic). The flow-level scheduler of each slice decides which user flows to serve within that slice using the resources allocated by the slice-level scheduler, and is typically opportunistic to time-varying channels.

Algorithm 1: GPS Slice-Level Scheduler.

```
Parameters: weight vector $\boldsymbol{w}$, number of resource blocks per slot $\mathtt{RB}$
for $t = 0, 1, 2, \cdots$ do
    For any slice $c \in [s]$ which is non-idle, allocate $(\gamma \cdot \mathtt{RB})$ resource blocks (with proper discretization) where
        $$\gamma = \frac{w_c}{\sum_{c' \in [s]} \mathbb{1}\{c' \text{ is non-idle}\} w_{c'}}$$
    Do the previous step on remaining unused resource blocks (some slices may not exhaust allocated blocks), until all the blocks are used or all packets are transmitted.
```

In this paper, we consider a parameterized ***S****lice-****L****evel* ***S****cheduler*, denoted by $\mathsf{SLS}(\boldsymbol{w})$. The vector $\boldsymbol{w} = (w_j)_{j \in [s]}$ indicates the weights of slices for resource allocation. Roughly speaking, w_j determines in some sense the proportion of resources that is to be allocated to jth slice. A larger w_j implies users in $\mathcal{U}_j$ are allocated more resources. Here, we present a simple *Generalized Processor Sharing* (GPS) slice-level scheduler in Algorithm 1 as a driving example.

Isolating Slices through Share Guarantees: Without loss of generality, we assume $\boldsymbol{w} \in \mathcal{W} := \{\boldsymbol{w}' \in [0, 1]^s : |\boldsymbol{w}'|_1 = 1, \boldsymbol{w}' \succeq \boldsymbol{\zeta}\}$ where $\boldsymbol{\zeta} \in [0, 1]^s$ indicates a pre-specified system constraint on the lowest resource allocation for each class, and thus provides a natural way to encode protection (guarantees on resource share) to each slice.

Remark 1 (*General Isolation Constraints for Slices*)**.** We let $\mathcal{W}$ to have a simplex shape under the constraint $\boldsymbol{w} \succeq \boldsymbol{\zeta}$. This is to simplify the tree-partitioning procedure which will be discussed in Section 2.6. In general, more complex isolation constraints (e.g. guarantees on the sum of weights of slices for a service controlling multiple slices) can be considered; we skip details for ease of exposition.

We assume jth slice ($j \in [s]$) pre-specifies its own ***F****low-****L****evel* ***S****cheduler*, denoted by FLS_j, for its specific needs. The scheduler FLS_j is channel- and queue-aware (for example, a MaxWeight scheduler).

Denote the parameterized ***H****ierarchical* ***S****cheduler* as $\mathsf{HS}(\boldsymbol{w}) := (\mathsf{SLS}(\boldsymbol{w}), (\mathsf{FLS}_j)_{j \in [s]})$. The class of schedulers $\{\mathsf{HS}(\boldsymbol{w}) : \boldsymbol{w} \in \mathcal{W}\}$ induces a capacity region $\mathcal{K}_{\mathcal{W}} \subset \mathcal{K}$, i.e, for any $\boldsymbol{\lambda} \in (\mathcal{K}_{\mathcal{W}})^{\circ}$ there exists $\boldsymbol{w} \in \mathcal{W}$ such that $\mathsf{HS}(\boldsymbol{w})$ stabilizes the system. We assume a fixed $\boldsymbol{\lambda} \in (\mathcal{K}_{\mathcal{W}})^{\circ}$ for the rest of this paper. The goal is to find the best parameter $\boldsymbol{w}$ that generates the largest rate of rewards, where the rewards are defined through regenerative cycles which will be discussed later. We denote $\mathcal{W}(\boldsymbol{\lambda}) \subset \mathcal{W}$ as the set of weights such that for any $\boldsymbol{w} \in \mathcal{W}(\boldsymbol{\lambda})$, $\mathsf{HS}(\boldsymbol{w})$ stabilizes the system.

Remark 2. In this work, we fix the flow-level schedulers (which can be seen as good state-of-the-art approaches, or may be independently decided by the tenant "owning" the slice) but optimize the slice-level scheduling to reduce the learning complexity. We believe this is a practical approach to adopt. It is worth noting that the joint optimization of both slice- and flow-level schedulers is an interesting direction to further investigate. If the flow-level scheduler can be well parameterized as was done for the slice-level counterpart, one could theoretically still approach the problem based on the same bandit framework proposed in this paper, but this would introduce a higher complexity for the parameter space and may result in longer convergence time. Other ideas for joint optimization can also be considered, e.g., multi-level bandit frameworks, but this is beyond the scope of this paper.

[3] If $\mathcal{U}^{\text{nb}} = \phi$, the system is such that there are no queueing stability concerns, which makes the problem simpler.

2.3. Reward model and regenerative dynamics

A queueing system with stochastic arrivals and departures consists of alternate idle and busy periods, i.e., periods of the system without/with packets.[4] A consecutive (idle + busy period) is called a *cycle*.[5] Next, we describe the system dynamics through such cycles before introducing the adaptive scheduler model in the next sub-section. We mostly follow assumptions from Section II-B of [28].

If $\mathsf{HS}(\boldsymbol{w})$ is implemented for the nth cycle, suppose that the system observes a *cycle length* denoted by $C^{(\boldsymbol{w})}(n)$ (possibly-infinite if $\mathsf{HS}(\boldsymbol{w})$ is unstable), and receives a sequence of *non-negative* rewards $(U^{(\boldsymbol{w})}(n,i))_{1\le i\le C^{(\boldsymbol{w})}(n)}$. Denote by $U^{(\boldsymbol{w})}(n)$ the *cycle reward* of the nth cycle, i.e., $U^{(\boldsymbol{w})}(n)$ equals the sum of $U^{(\boldsymbol{w})}(n,i)$ for $1 \le i \le C^{(\boldsymbol{w})}(n)$. We make the following model assumptions on the process $((C^{(\boldsymbol{w})}(n), U^{(\boldsymbol{w})}(n)))_{n\ge 1}$:

(1) The cycle length variables $C^{(\boldsymbol{w})}(n)$ are *i.i.d.* over n. This is automatically true due to the *i.i.d.* arrival/service model in Section 2.1 if $\mathsf{HS}(\boldsymbol{w})$ is Markovian, i.e., making decisions solely based on current system states. If $\mathsf{HS}(\boldsymbol{w})$ leverages past information (e.g., a proportionally-fair flow-level scheduler), we require the past information is cleared before a new cycle — note that this is solely an implementation choice so as to ensure that the rewards of each cycle are $\boldsymbol{w}$-conditionally independent, otherwise a fair comparison of different weight choices would not be possible.

(2) The cycle rewards $U^{(\boldsymbol{w})}(n)$ are also *i.i.d.* for $n \ge 1$. In addition, we assume that for any $\boldsymbol{w} \in \mathcal{W}$, $0 \le U^{(\boldsymbol{w})}(n) \le \bar{r}C^{(\boldsymbol{w})}(n)$ for some $\bar{r} > 0$, i.e., the cumulative rewards generated by $\mathsf{HS}(\boldsymbol{w})$ do not grow faster than linearly over time.

Remark 3 (*Reward Model*)**.** A wide range of user requirements or performance metrics can be captured with our reward model. For instance, suppose each packet is associated a bounded reward (e.g., over [0, 1]) upon reception, and the cycle reward $U^{(\boldsymbol{w})}(n)$ equals the sum of the packet rewards delivered within the nth cycle under scheduler $\mathsf{HS}(\boldsymbol{w})$. For a latency-sensitive user with strict packet deadlines, each packet reward can be set as 1 **only** if it meets the deadline; for a user that prefers a smaller mean packet delay, the per packet reward may be set as $(1 - c \cdot \texttt{delay}, 0)^+$ for a proper coefficient c. It is worth noting that the manner in which rewards are calculated/defined is not necessarily known by the base station in our model, which allows for very general (possibly user-customized) reward structures.

Finally, we assume that for any stable weight $\boldsymbol{w} \in \mathcal{W}(\boldsymbol{\lambda})$, cycle length $C^{(\boldsymbol{w})}(n)$ and thereby $U^{(\boldsymbol{w})}(n)$ are sub-exponentially-distributed.

Assumption 1 (*Cycle Length Distribution*)**.** If $\boldsymbol{w} \in \mathcal{W}(\boldsymbol{\lambda})$, $C^{(\boldsymbol{w})}(n)$ is a sub-exponential random variable. This implies that, there exist ($\boldsymbol{\lambda}$-dependent) parameters $(\xi_{\boldsymbol{w}}^2, \alpha_{\boldsymbol{w}})$, such that for all $n \ge 1$,

$$\mathbb{P}(|C^{(\boldsymbol{w})}(n) - \mathbb{E}[C^{(\boldsymbol{w})}(n)]| \ge \varepsilon) \le \begin{cases} 2e^{-\varepsilon^2/(2\xi_{\boldsymbol{w}}^2)} & 0 < \varepsilon \le \frac{\xi_{\boldsymbol{w}}^2}{\alpha_{\boldsymbol{w}}}, \\ 2e^{-\varepsilon/(2\alpha_{\boldsymbol{w}})} & \varepsilon > \frac{\xi_{\boldsymbol{w}}^2}{\alpha_{\boldsymbol{w}}}. \end{cases} \tag{1}$$

As pointed out in [28], this assumption holds true if the system has bounded arrival and channel distributions, and the policies considered are Markovian, which follows an argument of [29].

2.4. Adaptive hierarchical scheduler, clipping and feedback

We are interested in designing an online algorithm that learns the optimal weight parameter inducing the best rate of rewards. For this purpose, we model the problem as a multi-armed bandit (operating over cycles) and each choice of parameters as an arm (the arm set is continuous). At each cycle n, the bandit algorithm, referred to as the ***A****daptive* ***H****ierarchical* ***S****cheduler* (AHS), chooses a weight $\boldsymbol{w} \in \mathcal{W}$, (action of bandit from a continuum) based on past rewards and collects rewards using the Hierarchical Scheduler $\mathsf{HS}(\boldsymbol{w})$.

Clipping and Motivation: As in [28], the AHS makes decisions to change the weight $\boldsymbol{w} \in \mathcal{W}$ only at the end of queueing cycles (i.e. when the system is empty for users with finite backlogs). It can also choose to terminate a queueing cycle while it is progressing, meaning discard all packets and force the system to become empty. We refer to this operation as *clipping a cycle*. Clipping is used to ensure that a "bad" weight choice (e.g. when the resources allocated to a slice are too small and lead to queue instability within the slice) does not result in arbitrarily long cycles. Furthermore, rewards from a clipped cycle should be properly penalized, ensuring that bad weight choices are eliminated as the algorithm adapts over time.

We represent AHS by the sequence $\pi = (\pi_n)_{n\ge 1}$, where $\pi_n = (\boldsymbol{W}_n^\pi, L_n^\pi) \in \mathcal{W} \times (\mathbb{Z}_+ \cup \{+\infty\})$. A decision π_n consists of selecting both a weight parameter and a clipping threshold. In other words, in the nth cycle, if $\pi_n = (\boldsymbol{w}, l)$, then AHS will use $\mathsf{HS}(\boldsymbol{w})$ to collect rewards, and will clip the cycle if its duration exceeds l time-slots. The stochastic feedback that is received by AHS at the end of the nth cycle is represented by Z_n^π. Formally,

$$Z_n^\pi = (\hat{C}_n^\pi,\ \hat{U}_n^\pi,\ \mathbb{1}_{\{C^{(\boldsymbol{w})}(n) > l\}}),$$

[4] To be precise, a system is idle when there are no packets *excluding infinitely-backlogged queues*. Without loss of generality, we assume arrivals occur at the start of a slot while departures occur at the end.

[5] A weaker notion will be introduced in Remark 7 (Section 4) for practical use in high-load systems, where the queues may not be strictly idle to restart a cycle.

where $\hat{C}^{\pi}_{n}$ and $\hat{U}^{\pi}_{n}$ denote the (random) observed length and reward for the nth cycle, i.e., $\hat{C}^{\pi}_{n} = \min(C^{(\boldsymbol{w})}(n), l)$ and $\hat{U}^{\pi}_{n} = \sum_{j=1}^{\hat{C}^{\pi}_{n}} U^{(\boldsymbol{w})}(n, j)$.

Remark 4 (*Soft Clipping*). It is worth noting that the clipping discussed above is an implementation choice (rather than a necessary assumption) which we use to simplify the interpretation of the framework. For some applications it may be unacceptable to drop packets. In this case, it is possible to use an alternative soft clipping strategy as follows: whenever a cycle is clipped, instead of dropping packets, a default stabilizing policy $\mathsf{HS}(\boldsymbol{w}_0)$ is implemented until the system becomes idle again (we assume some $\boldsymbol{w}_0 \in \mathcal{W}(\boldsymbol{\lambda})$ is known a priori). By applying Theorem 2.3 of [29], it can be proven that the time taken by each queue-clearing process is polynomial to the total length of queues at clipping (which is at the same order of the clipping threshold). As we will show in the next section, the threshold only grows logarithmically, therefore, the extra reward loss resulting from this process is poly-logarithmic. Orderwise, this does not affect the main theoretical result in Section 3.4.

2.5. Reward rates, optimal weight and regret

As discussed above, the goal of AHS is to locate the optimal weight parameter leading to the highest rate of rewards, which will be formally-defined in this section.

First, for AHS (denoted by) π, denote by $M_{\pi}(n)$ the total number of slots for first n (possibly-clipped) cycles and by $N_{\pi}[\tau]$ the number of *completed* cycles before time τ, i.e., $N_{\pi}[\tau] = \max(n : M_{\pi}(n) \leq \tau)$. Let $\mathsf{Rew}_{\pi}[\tau]$ be the cumulative rewards collected under π over the first τ slots, i.e.,

$$\mathsf{Rew}_{\pi}[\tau] := \sum_{n=1}^{N_{\pi}[\tau]} \hat{U}^{\pi}_{n} + \tilde{U}(\tau),$$

where $\tilde{U}(\tau)$ is the shorthand notation for the sum of rewards of the first $\tau - M_{\pi}(N_{\pi}[\tau])$ slots of the $(N_{\pi}[\tau]+1)$-th cycle.

For any $\boldsymbol{w} \in \mathcal{W}$, we define a static non-clipping policy $\pi^{(\boldsymbol{w})}$ such that $\pi^{(\boldsymbol{w})}_{n} = (\boldsymbol{w}, \infty)$ for any $n \geq 1$. Then we can define a reward rate function f where

$$f(\boldsymbol{w}) = \limsup_{\tau\to\infty} \frac{1}{\tau}\mathbb{E}[\mathsf{Rew}_{\pi^{(\boldsymbol{w})}}[\tau]].$$

This function f indeed maps each weight choice to the rate of rewards generated by $\mathsf{HS}(\boldsymbol{w})$, and is to be optimized. Note that by Renewal Theory, for any stable weight $\boldsymbol{w} \in \mathcal{W}(\boldsymbol{\lambda})$,

$$f(\boldsymbol{w}) = \frac{\mathbb{E}[U^{(\boldsymbol{w})}(1)]}{\mathbb{E}[C^{(\boldsymbol{w})}(1)]}. \tag{2}$$

This motivates us to use (empirical reward/empirical length), or the empirical reward rate, as the estimator of f.

Denote the optimal weight by $\boldsymbol{w}^* := \operatorname{argmax}_{\boldsymbol{w}\in\mathcal{W}} f(\boldsymbol{w})$ and the optimal rate of rewards as $f^* := f(\boldsymbol{w}^*)$. For the analysis of our algorithm, we will assume $\boldsymbol{w}^*$ to be unique and lie in the stable weight region $\mathcal{W}(\boldsymbol{\lambda})$. In practice, an unstable system is typically unwelcome and should be penalized in rewards, which makes this assumption reasonable.

Finally, we define the cumulative regret of AHS π with respect to the best static non-clipping Hierarchical Scheduler, i.e., $\pi^{(\boldsymbol{w}^*)} = (\boldsymbol{w}^*, \infty)_{n\geq 1}$, as follows,

$$\mathsf{Reg}_{\pi}[\tau] = \mathbb{E}[\mathsf{Rew}_{\pi^{(\boldsymbol{w}^*)}}[\tau] - \mathsf{Rew}_{\pi}[\tau]].$$

2.6. Tree-based partitioning and structural assumption

In this paper we will present a tree search-type algorithm for the blackbox optimization of f. Let us first define a tree-based partitioning on $\mathcal{W}$ and a joint structural assumption on f and the partitions. This is a standard approach adopted in recent studies on bandit algorithms with a continuous arm set.

Assume that $\mathcal{W}$ is partitioned into an infinite binary tree $\mathcal{T} = \{(h, i)_{h\geq 0, 1\leq i\leq 2^h}\}$. The index pair (h, i) represents the ith node at the depth h of the tree, which is associated with a region $\mathcal{P}_{h,i} \subset \mathcal{W}$. The collection $(\mathcal{P}_{h,i})_{h\geq 0, 1\leq i\leq 2^h}$ is referred to as a "tree of coverings", which satisfies $\mathcal{P}_{0,1} = \mathcal{W}$ and $\mathcal{P}_{h,i} = \mathcal{P}_{h+1,2i-1} \cup \mathcal{P}_{h+1,2i}$ for all $(h, i) \in \mathcal{T}$. In this paper, we will use a "normal" partition: The coverings $(\mathcal{P}_{h,i})_{h\geq 0, 1\leq i\leq 2^h}$ are determined by sub-routine Split exhibited in Algorithm 2, i.e., starting from the root $\mathcal{W}$, iteratively halving each node simplex into two child simplices.

Let $f^*_{h,i} = \sup_{\boldsymbol{w}\in\mathcal{P}_{h,i}} f(\boldsymbol{w})$. The sub-optimality of node (h, i) is denoted by $\Delta_{h,i} := f^* - f^*_{h,i}$. The unique node at depth h that contains $\boldsymbol{w}^*$ is denoted by (h, i^*_h), i.e., $f^*_{h,i^*_h} = f^*$. For purposes of performing regret analysis, we will make the following assumption on function smoothness of f with respect to $\mathcal{W}$. This assumption follows that used in [20].

Assumption 2 (*Function Smoothness*). There exists $\nu > 0$ and $\rho \in (0, 1)$ such that the following holds: If a node (h, i) is such that $\Delta_{h,i} \leq c\nu\rho^h$ for some constant $c \geq 0$, then $f(\boldsymbol{w}) > f^* - \max\{2c, c + 1\}\nu\rho^h$ for all $\boldsymbol{w} \in \mathcal{P}_{h,i}$.

Algorithm 2: Subroutines for Tree Partitioning.

```
procedure SPLIT($\mathcal{P}_{h,i}$)
    $(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_s) \leftarrow \mathcal{P}_{h,i}$        ▷ Retrieving vertices of $\mathcal{P}_{h,i}$
    $(j, k) \leftarrow \mathrm{argmax}_{(j',k')} ||\boldsymbol{v}_{j'} - \boldsymbol{v}_{k'}||_2$
    $\boldsymbol{v}' \leftarrow (\boldsymbol{v}_j + \boldsymbol{v}_k)/2$
    $\mathcal{P}_{h+1,2i-1} \leftarrow$ Replace vertex $\boldsymbol{v}_j$ of $\mathcal{P}_{h,i}$ by $\boldsymbol{v}'$
    $\mathcal{P}_{h+1,2i} \leftarrow$ Replace vertex $\boldsymbol{v}_k$ of $\mathcal{P}_{h,i}$ by $\boldsymbol{v}'$
    return $\mathcal{P}_{h+1,2i-1}, \mathcal{P}_{h+1,2i}$
procedure SELECT($\mathcal{P}_{h,i}$)
    $(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_s) \leftarrow \mathcal{P}_{h,i}$
    return $\boldsymbol{w}_{h,i} := (1/s) \sum_{j=1}^{s} \boldsymbol{v}_s$
```

This assumption implies that, for any optimal node (h, i_h^*), $f^* - \inf_{\boldsymbol{w} \in \mathcal{P}_{h,i_h^*}} f(\boldsymbol{w}) < \nu\rho^h$ for some μ and ρ, i.e., the worst loss in reward rates of an optimal node contracts geometrically with h. A related notion is *near-optimality dimension* with respect to (ν, ρ). Here we use the definition originally stated in [19].

Definition 1. The near-optimality dimension of function f with respect to parameters (ν, ρ) is given by $d(\nu, \rho) := \inf\{d' \in \mathbb{R}_+ : \exists \mathrm{C}(\nu, \rho), \forall h \geq 0, \mathcal{N}_h(2\nu\rho^h) \leq \mathrm{C}(\nu, \rho)\rho^{-d'h}\}$, where $\mathcal{N}_h(\epsilon)$ is the number of nodes (h, i) such that $f_{h,i}^* \geq f^* - \epsilon$.

Roughly speaking, $d(\nu, \rho)$ measures the difficulty of the problem: The larger the dimension is, the larger is the number of "near-optimal" nodes which are hard to distinguish from the optimal node, implying that more exploration is needed. This notion will be used in regret analysis.

3. Algorithm design and analysis

In this section, we introduce our main result — ***C**ycle-based **HOO** with **C**lipping (CHOOC)*, an HOO-type bandit algorithm to determine the optimal weight parameter $\boldsymbol{w}^*$. Compared with the original HOO in [4], we need to address two main challenges. First, each action (arm decision) is fixed over an entire (stochastic) cycle time, and the function to be optimized is a ratio of the cumulative reward across cycles divided by the cumulative cycle time. Thus, the exploration bonus used for the upper confidence bound has to account for the double stochasticity in both rewards and durations. Second, the distributions of cycle variables cannot be described by homogeneous sub-Gaussian parameters; instead, the length (and reward accordingly) of a cycle can be as large as infinite for some weight choices, suggesting that a proper cycle clipping rule is necessary (which does not drop an excessive number of packets).

As discussed in the *Related Work*, a similar bandit problem setting was tackled in [28], but with a finite (discrete) set of arms. In that setting, the arms that generate 'short' cycles (whose lengths are exponentially distributed under specific parameters) can be differentiated from those generating 'long' cycles by designing an appropriate exploration bonus, and thus the regrets can simply be computed separately in the analysis. In our work, as we have an infinite collection of arms (more precisely a continuum of arms), the discretization of arms via a tree-like structure complicates the design of an exploration bonus (due to the continuity of the reward ratio) and the aforementioned separation in regret analysis (since there is no clear border between stable and unstable weights). Thus, additional effort is needed to ensure that the algorithm still yields a logarithmic regret with cycles and clipping. In the following, we will present our proposed algorithm and introduce several key assumptions and design elements.

3.1. Hyper-parameters and algorithm design

Let us first discuss necessary hyper-parameters to be used in our algorithm. Before that, denote $\mathcal{W}(\boldsymbol{\lambda}; \xi^2, \alpha) \subset \mathcal{W}(\boldsymbol{\lambda})$ as the largest set of arms such that for any $\boldsymbol{w} \in \mathcal{W}(\boldsymbol{\lambda}; \xi^2, \alpha)$, variables $C^{(\boldsymbol{w})}(1)$ and $U^{(\boldsymbol{w})}(1)$ are both (ξ^2, α)-sub-exponentially distributed.

Assumption 3 (*Hyper-parameters*)**.** We assume that the hyper-parameters are chosen to satisfy the following conditions:
(a) (ν, ρ) that satisfies the condition in Assumption 2.
(b) (ξ^2, α) and $z \geq 1$ such that $\mathcal{P}_{z,i_z^*} \subset \mathcal{W}(\boldsymbol{\lambda}; \xi^2, \alpha)$,
(c) $\mu_{\min}, \mu_{\max}$ such that for any $\boldsymbol{w} \in \mathcal{P}_{z,i_z^*}$, $\mu_{\min} \leq \mathbb{E}[C^{(\boldsymbol{w})}(1)] \leq \mu_{\max}$,
(d) $r_{\max}$ such that $\mathbb{E}[U^{(\boldsymbol{w})}(1)|C^{(\boldsymbol{w})}(1) = l] \leq r_{\max}\, l$ for all $\boldsymbol{w} \in \mathcal{P}_{z,i_z^*}$ and $l \geq 1$.

Remark 5. We assume a depth z is known such that the optimal node (z, i_z^*) and its descendants are purely (ξ^2, α)-sub-exponential (in terms of corresponding cycle length/reward variables). A "pure" node provides necessary concentration properties such that the exploration bonus, a term we will formally discuss later, sufficiently compensates the empirical rewards used to approximate f, which may be under-performing due to the stochasticity of feedback. This condition can be met by setting z to be large enough.

With the exception of z, the other parameters described above are used for defining the exploration bonus. Parameters of type (a) and (b) are standard in HOO-type algorithms. Parameters similar to those in (c) and (d) are commonly used in bandit algorithms where each action costs non-unit amount of resources (cycle time in our model), e.g., [27,30]. Note that $f^* \leq r_{\max} \leq \bar{r}$ (see Section 2.3 and Eq. (2)). While Assumption 3 is needed for our regret analysis, it is worth noting that none of the assumptions are "hard constraints"; indeed empirically selecting those parameters (instead of formally verifying that the conditions are satisfied) is sufficient in practice, as we observe good robust performance in our simulations. We refer to Section 4 (in particular, Remark 6) for details.

Algorithm 3: Cycle-Based HOO with Clipping (CHOOC)

```
1:  Inputs: Schedulers HS(·), Set of Available Weights $\mathcal{W}$, Tree Partitioning Subroutines Split and Select
2:  Hyper-parameters: Smoothness Parameters: $(\nu, \rho)$, Sub-Exponential Parameters: $(\xi^2, \alpha)$, Initial Depth $z$, Other
    Parameters: $\mu_{\min}, r_{\max}, \beta, \kappa$ $(\beta > \max(2\mu_{\max}, \mu_{\max}+\xi^2/\alpha), \kappa > 4\alpha)$
3:  Initialization: $\mathcal{P}_{0,1} \leftarrow \mathcal{W}$, then partition the arm space hierarchically until depth $z$:
4:      for $0 \leq h \leq z-1$ do
5:          for $1 \leq i \leq 2^h$ do
6:              $\mathcal{P}_{h+1,2i-1}, \mathcal{P}_{h+1,2i} \leftarrow$ Split$(\mathcal{P}_{h,i})$
7:      Establish a (virtual) node $(z-1, *)$ as the parent of nodes $\{(z,i) : 1 \leq i \leq 2^z\}$
8:      $\mathcal{T}_{\text{exp}} \leftarrow \{(z-1,*)\}$, $B_{z,i} \leftarrow \infty$, $T_{z,i} \leftarrow 0 \, \forall 1 \leq i \leq 2^z$
9:  for cycle index $n = 1, 2, \cdots$ do
10:     $(h,i) \leftarrow (z-1,*)$, $P \leftarrow \phi$
11:     while $(h,i) \in \mathcal{T}_{\text{exp}}$ do
12:         if $h = z-1$ then
13:             $j \leftarrow \text{argmax}_{1 \leq j \leq 2^z} B_{z,j}$ (with a tie-breaker)
14:             $(h,i) \leftarrow (z,j)$
15:         else
16:             $j \leftarrow \text{argmax}_{j \in \{0,1\}} B_{h+1,2i-j}$ (w/ tie-breaker)
17:             $(h,i) \leftarrow (h+1, 2i-j)$
18:         $P \leftarrow P \cup \{(h,i)\}$
19:     $(H,I) \leftarrow (h,i)$, $\mathcal{T}_{\text{exp}} \leftarrow \mathcal{T}_{\text{exp}} \cup \{(H,I)\}$
20:     $\boldsymbol{w} \leftarrow$ Select$(\mathcal{P}_{H,I})$, $l \leftarrow \beta + \kappa \log n$
21:     Run $n$-th cycle using HS$(\boldsymbol{w})$ with threshold $l$.
22:     Observe cycle length/reward $\hat{C}$ and $\hat{U}$.
23:     for all $(h,i) \in P$ do
24:         $T_{h,i} \leftarrow T_{h,i} + 1$
25:         $\hat{\mu}^{\text{C}}_{h,i} \leftarrow (1 - 1/T_{h,i})\hat{\mu}^{\text{C}}_{h,i} + \hat{C}/T_{h,i}$
26:         $\hat{\mu}^{\text{U}}_{h,i} \leftarrow (1 - 1/T_{h,i})\hat{\mu}^{\text{U}}_{h,i} + \hat{U}/T_{h,i}$
27:         $\hat{R}_{h,i} \leftarrow \hat{\mu}^{\text{U}}_{h,i}/\hat{\mu}^{\text{C}}_{h,i}$
28:     $\mathcal{P}_{H+1,2I-1}, \mathcal{P}_{H+1,2I} \leftarrow$ Split$(\mathcal{P}_{H,I})$
29:     For $j = \{0,1\}$, $B_{H+1,2I-j} \leftarrow \infty$, $T_{H+1,2I-j} \leftarrow 0$
30:     Backward update from leaves to the root in $\mathcal{T}_{\text{exp}}$:
31:         $\Phi_{h,i} \leftarrow$ Compute according to (3)
                                                        ▷ specially-designed exploration bonus
32:         $B_{h,i} \leftarrow \hat{R}_{h,i} + \Phi_{h,i} + \nu\rho^h$
33:         $B_{h,i} \leftarrow \min(B_{h,i}, \max(B_{h+1,2i-1}, B_{h+1,2i}))$
```

The full algorithm is presented in Algorithm 3. In the sequel, the term "explore node (h,i)" means "select a representative weight $\boldsymbol{w}_{h,i}$ inside $\mathcal{P}_{h,i}$ and run HS$(\boldsymbol{w}_{h,i})$", where $\boldsymbol{w}_{h,i}$ is determined by subroutine Select described in Algorithm 2. Each node is associated with an *upper confidence bound* (or "B-value"), which is an optimistic estimate of $f^*_{h,i}$ that equals the corresponding empirical reward rate plus an exploration bonus, both to be defined later.

The AHS first partitions $\mathcal{W}$ into $(\mathcal{P}_{z,i})_{1 \leq i \leq 2^z}$, and initializes the B-values of depth-z's nodes as ∞ (line 3–8). The algorithm starts by exploring nodes at depth z. This is to guarantee that the selected weights for the optimal nodes always induce (ξ^2, α)-sub-exponential cycles and thus, the optimal nodes (of depth $\geq z$) are associated with well-behaved B-values as commented in Remark 5, which is essential to the theoretical success of the algorithm.

The AHS starts with no explored nodes (except the virtual parent $(z-1,*)$) in $\mathcal{T}_{\text{exp}}$ (the "explored tree"). Before each cycle (line 10–18), the AHS traverses $\mathcal{T}_{\text{exp}}$ along a path P in which each node has the better B-value among its siblings. This procedure ends until an unexplored node is reached. This node will be explored for this cycle with $\mathcal{T}_{\text{exp}}$ expanded accordingly (line 19–20).

After a cycle ends, the AHS collects feedback and updates the empirical reward and cycle length for each node in P (line 23–26). The empirical reward rate is the quotient of the two (line 27), aligned with Eq. (2). Finally (line 30–33), for all nodes (h, i) in $\mathcal{T}_{\text{exp}}$, new exploration bonuses $(\Phi_{h,i}+\nu\rho^h)$ are computed and B-values are updated, which can be further refined using the fact that the optimistic estimate of a node is no larger than the maximal estimate of its children.

3.2. Clipping

It is essential to design a proper clipping rule such that all the "bad" weights leading to queue instability are penalized without blocking the learning process while "good" weights are preserved with little clipping. In our algorithm, we apply the clipping threshold $l_n = \beta + \kappa \log n$ for nth cycle, where we require $\kappa > 4\alpha$ and β reasonably large. This rule guarantees that any unstable cycle is clipped while cycles induced by $\boldsymbol{w}^*$ (and its neighborhood) are rarely affected, implying negligible performance loss. Indeed, by the property in (1), we have that for any $\boldsymbol{w} \in \mathcal{W}(\boldsymbol{\lambda}; \xi^2, \alpha)$,

$$\mathbb{E}[\sum_{n=1}^{\infty} \mathbb{1}_{\{C^{(\boldsymbol{w})}(n)>l_n\}}] \leq C_0 \text{ (a constant).}$$

Instead, a constant threshold rule inevitably results in a linear number of clippings.

Denote by $\hat{C}^{(\boldsymbol{w},l)}(n)$, $\hat{U}^{(\boldsymbol{w},l)}(n)$ the *observed* cycle variables given $\pi_n = (\boldsymbol{w}, l)$. For the purpose of regret analysis, we set the initial threshold β to be large enough: $\beta > \max(2\mu_{\max}, \mu_{\max}+\xi^2/\alpha)$. In addition, we make the following technical assumptions on cycle clipping.

Assumption 4 (*Technical Assumptions*)**.** (a) For any $\boldsymbol{w} \in \mathcal{W} \setminus \mathcal{W}(\boldsymbol{\lambda}; \xi^2, \alpha)$, $\sup_{l\geq\beta} \frac{\mathbb{E}[\hat{U}^{(\boldsymbol{w},l)}(1)]}{\mathbb{E}[\hat{C}^{(\boldsymbol{w},l)}(1)]} \leq f^* - \delta$ for some $\delta > 0$.

(b) For any $\boldsymbol{w} \in \mathcal{P}_{z,i_z^*}$, the l-interrupted cycle reward $\hat{U}^{(\boldsymbol{w},l)}(1)$ is (ξ^2, α)-sub-exponential for all $l \geq \beta$.

Assumption 4(a) says that for any $\boldsymbol{w} \in \mathcal{W} \setminus \mathcal{W}(\boldsymbol{\lambda}; \xi^2, \alpha)$, where the effect of clipping is *not* negligible, clipping does not significantly boost the reward rate to exceed f^*. The underlying intuition is that an unstable system typically receives much poorer rewards[6]. Assumption 4(b) requires that the clipping does not hurt the sub-exponential property of reward variables for $\boldsymbol{w} \in \mathcal{P}_{z,i_z^*}$. Both (a),(b) can be met if the initial threshold β is sufficiently large, since as $l \to \infty$, we have $\mathbb{E}[\hat{U}^{(\boldsymbol{w},l)}(1)]/\mathbb{E}[\hat{C}^{(\boldsymbol{w},l)}(1)] \to f(\boldsymbol{w})$ and $\hat{U}^{(\boldsymbol{w},l)}(1) \to \hat{U}^{(\boldsymbol{w})}(1)$ *a.s.*

3.3. Upper confidence bound

For a better description, denote (H_n, I_n), $(\hat{C}_n, \hat{U}_n)$, (C_n, U_n) as the selected node, the observed length/reward and the unclipped length/reward for cycle n. Let $\mathcal{D}(h, i)$ be the descendants of (h, i) including itself. The number of samples of (h, i) for first n cycles is thereby $T_{h,i}(n) = \sum_{t=1}^{n} \mathbb{1}_{\{(H_t,I_t)\in\mathcal{D}(h,i)\}}$. The empirical reward rate of (h, i), once explored, is given by

$$\hat{R}_{h,i}(n) := \frac{\hat{\mu}^{\mathrm{U}}_{h,i}(n)}{\hat{\mu}^{\mathrm{C}}_{h,i}(n)} = \frac{(1/T_{h,i}(n))\sum_{t=1}^{n} \hat{U}_t \mathbb{1}_{\{(H_t,I_t)\in\mathcal{D}(h,i)\}}}{(1/T_{h,i}(n))\sum_{t=1}^{n} \hat{C}_t \mathbb{1}_{\{(H_t,I_t)\in\mathcal{D}(h,i)\}}}.$$

To ensure sufficient exploration of various weights, the upper confidence bound used for comparing nodes' performance is defined as $B_{h,i}(n) := \hat{R}_{h,i}(n) + \nu\rho^h + \Phi_{h,i}(n)$,[7] where

$$\Phi_{h,i}(n) = \frac{(1+r_{\max})\epsilon_{h,i}(n) + \epsilon'_{h,i}(n)}{\mu_{\min} + \epsilon_{h,i}(n)}, \tag{3}$$

and

$$\epsilon_{h,i}(n) = \begin{cases} \sqrt{\frac{8\xi^2 \log n}{T_{h,i}(n)}} & \sqrt{\frac{8\xi^2 \log n}{T_{h,i}(n)}} \leq \frac{\xi^2}{\alpha}, \\ \frac{8\alpha \log n}{T_{h,i}(n)} & \text{otherwise,} \end{cases}$$

$$\epsilon'_{h,i}(n) = \frac{\pi^2}{6} r_{\max} \frac{(\beta+2\alpha+\kappa \log T_{h,i}(n)+1)}{T_{h,i}(n)} e^{-\beta/4\alpha}.$$

The term $(\nu\rho^h + \Phi_{h,i}(n))$ is referred to as the exploration bonus. Paralleling the HOO algorithm, the exploration bonus is designed such that the optimal nodes of any depth (almost) always have a sufficiently optimistic estimate, which is a key to the success of UCB-type algorithms. Unlike HOO, however, additional effort is exerted to account for the fact that $\hat{R}_{h,i}(n)$ is a ratio of two random variables (i.e., to deal with the concentration of the empirical reward rate rather than the reward itself) as well as the clipping error. We formally present the discussion in the Lemma below.

[6] In practice, one can simply set the reward for any clipped cycle to be 0.

[7] We do not consider the last-step refinement of $B_{h,i}(n)$, since it does not affect the correctness of the regret analysis.

Lemma 1. *For all optimal nodes (h, i^*_h), $h \geq z$, and all $n \geq 1$, we have that $\mathbb{P}\left(B_{h,i^*_h}(n) \leq f^*\right) \leq 2n^{-3}$.*

Proof (*Proof Sketch*)**.** Let i^* be the shorthand of i^*_h when there is no ambiguity.

The first part of the bonus, $\nu\rho^h$, compensates for the gap of $f^* - \inf_{\boldsymbol{w}\in\mathcal{P}_{h,i^*}} f(\boldsymbol{w})$. Suppose there is no stochasticity of feedback or clipping error, then $\hat{R}_{h,i}(n)$ is equivalent to:

$$\frac{(1/T_{h,i}(n))\sum_{t=1}^{n} \mathbb{E}[U_t]\mathbb{1}_{\{(H_t,I_t)\in\mathcal{D}(h,i)\}}}{(1/T_{h,i}(n))\sum_{t=1}^{n} \mathbb{E}[C_t]\mathbb{1}_{\{(H_t,I_t)\in\mathcal{D}(h,i)\}}},$$

i.e., replacing $\hat{U}_t, \hat{C}_t$ as in the original $\hat{R}_{h,i}(t)$ by $\mathbb{E}[U_t], \mathbb{E}[C_t]$. Let $\mu^{\mathtt{U},\dagger}_{h,i}(n)$ and $\mu^{\mathtt{C},\dagger}_{h,i}(n)$ denote the numerator and the denominator of the above expression. By (2) and the remark of Assumption 2, for any optimal node (h, i^*), we have that

$$\mu^{\mathtt{U},\dagger}_{h,i^*}(n)/\mu^{\mathtt{C},\dagger}_{h,i^*}(n) + \nu\rho^h > f^*.$$

The term $\Phi_{h,i^*}(n)$ is used to compensate for stochasticity of cycle rewards and lengths as well as clipping, i.e., the gap between $\hat{\mu}^{\mathtt{U}}_{h,i^*}(n)/\hat{\mu}^{\mathtt{C}}_{h,i^*}(n)$ and $\mu^{\mathtt{U},\dagger}_{h,i^*}(n)/\mu^{\mathtt{C},\dagger}_{h,i^*}(n)$. By manipulating terms, we have that

$$\frac{\mu^{\mathtt{U},\dagger}_{h,i^*}(n)}{\mu^{\mathtt{C},\dagger}_{h,i^*}(n)} - \frac{\mu^{\mathtt{U},\dagger}_{h,i^*}(n) - (\epsilon_{h,i^*}(n) + \epsilon'_{h,i^*}(n))}{\mu^{\mathtt{C},\dagger}_{h,i^*}(n) + \epsilon_{h,i^*}(n)} \leq \Phi_{h,i^*}(n).$$

Therefore, combining the discussions above, it follows that

$$\begin{aligned}&\mathbb{P}\left(\hat{R}_{h,i^*}(n) + \Phi_{h,i^*}(n) + \nu\rho^h \leq f^*\right)\\ &\quad\leq \mathbb{P}\left(\mu^{\mathtt{U},\dagger}_{h,i^*}(n) - \epsilon'_{h,i^*}(n) - \epsilon_{h,i^*}(n) \geq \hat{\mu}^{\mathtt{U}}_{h,i^*}(n)\right) + \mathbb{P}\left(\mu^{\mathtt{C},\dagger}_{h,i^*}(n) + \epsilon_{h,i^*}(n) \leq \mu^{\mathtt{C}}_{h,i^*}(n)\right).\end{aligned}$$

Here, term $\epsilon'_{h,i^*}(n)$ is used to rectify the clipping-induced gap $\mu^{\mathtt{U},\dagger}_{h,i^*}(n) - \hat{\mu}^{\mathtt{U},\dagger}_{h,i^*}(n)$.[8] By algebra, we can show that

$$\mu^{\mathtt{U},\dagger}_{h,i^*}(n) - \hat{\mu}^{\mathtt{U},\dagger}_{h,i^*}(n) \leq \epsilon'_{h,i^*}(n),$$

given that $\beta > \max(2\mu_{\max}, \mu_{\max}+\xi^2/\alpha)$. The term $\epsilon_{h,i^*}(n)$ serves as the high-probability bound of $|\hat{\mu}^{\mathtt{U}}_{h,i^*}(n) - \hat{\mu}^{\mathtt{U},\dagger}_{h,i^*}(n)|$ and its cycle length counterpart, following Azuma–Hoeffding inequality for Martingale differences (sub-exponential version), Assumption 3(b) and 4(b). This concludes the proof. □

3.4. Main theoretical result

Theorem 1. *Given that the hyper-parameters satisfy Assumption 3 and 4, the cumulative regret of π induced by Algorithm 3 has the following upper bound:*

$$Reg_{\pi}[\tau] \leq C^{Reg}\,(C(\nu,\rho))^{\frac{1}{d(\nu,\rho)+2}}\,\tau^{\frac{d(\nu,\rho)+1}{d(\nu,\rho)+2}}(\log\tau)^{\frac{1}{d(\nu,\rho)+2}} + \mathcal{O}(\log^4\tau)$$

for a universal constant C^{Reg}.

This result is of the same order of HOO in spite of the cyclic behavior of the algorithm and clipping. Before giving a proof sketch on the main theorem, let us introduce another key lemma, stating that the number of visits to "not-near-optimal" nodes is sub-linear. First, we define a modified reward rate function $\tilde{f}$ where

$$\tilde{f}(\boldsymbol{w}) = \begin{cases} f(\boldsymbol{w}) & \boldsymbol{w} \in (\mathcal{W}(\boldsymbol{\lambda}; \xi^2, \alpha))^{\circ},\\ \sup\limits_{l\geq\beta} \dfrac{\mathbb{E}[\hat{U}^{(\boldsymbol{w},l)}(1)]}{\mathbb{E}[\hat{C}^{(\boldsymbol{w},l)}(1)]} & \text{otherwise.}\end{cases}$$

The function value for $\boldsymbol{w} \notin (\mathcal{W}(\boldsymbol{\lambda}; \xi^2, \alpha))^{\circ}$ represents an upper bound on the reward rate *under clipping* (see Assumption 4(a)). The gap $\tilde{\Delta}_{h,i}$ is defined accordingly.

Lemma 2. *For all suboptimal nodes (h, i) such that $\tilde{\Delta}_{h,i} > \nu\rho^h$, we have that for all $n \geq 1$,*

$$\mathbb{E}[T_{h,i}(n)] = \begin{cases}\mathcal{O}(\log n/(\tilde{\Delta}_{h,i}-\nu\rho^h)^2) & \textit{if } (h,i)\in\mathcal{D}(z, {i_z}^*),\\ \mathcal{O}(\log^3 n/(\tilde{\Delta}_{h,i}-\nu\rho^h)^2) & \textit{otherwise.}\end{cases}$$

[8] The term $\hat{\mu}^{\mathtt{U},\dagger}_{h,i}(n)$ is defined as $\mu^{\mathtt{U},\dagger}_{h,i}(n)$ with U_t replaced by $\hat{U}_t$.

Proof (*Proof Sketch*)**.** By Lemma 14 of [4], we have that for any $u \geq 1$,

$$\mathbb{E}[T_{h,i}(n)] \leq u + \sum_{t=u+1}^{n} \mathbb{P}\big(\underbrace{\bigcup_{s=1}^{t} \{B_{s,i_s^*}(t) \leq f^*\}}_{\mathcal{E}_1(t)} \bigcup \underbrace{\{B_{h,i}(t) > f^*\} \cap \{T_{h,i}(t) > u\}}_{\mathcal{E}_2(t)}\big).$$

By Lemma 1, we have that $\sum_{t=1}^{n} \mathcal{E}_1(t) \leq \pi^2/6$. It suffices to show when $T_{h,i}(t) > u$ for a reasonably large u, the event $\{\hat{R}_{h,i}(t) + \Phi_{h,i}(t) > f^* - \nu\rho^h\}$ is unlikely to happen. For $(h, i) \in \mathcal{D}(z, i_z{}^*)$, the observed cycle length/reward variables are (ξ^2, α)-sub-exponential by Assumption 3(b). Accordingly, we can find a high-probability upper bound $f_{h,i}^* + \Phi'_{h,i}(t)$ for $\hat{R}_{h,i}(t)$, where the term $\Phi'_{h,i}(t)$ is similar to $\Phi_{h,i}(t)$ with a slight adjustment on the clipping, and both $\Phi'_{h,i}(t)$ and $\Phi_{h,i}(t)$ are of the order $\mathcal{O}(\sqrt{\log t/T_{h,i}(t)})$. This suggests that it suffices to find u such that $T_{h,i}(t) > u$ and $\Phi'_{h,i}(t) + \Phi_{h,i}(t) \leq f^* - f_{h,i}^* - \nu\rho^h = \Delta_{h,i} - \nu\rho^h$, which implies the constant $u \sim \mathcal{O}(\log t/(\Delta_{h,i} - \nu\rho^h)^2)$.

For $(h, i) \notin \mathcal{D}(z, i_z{}^*)$, a looser bound for $\Phi'_{h,i}(t)$ can be shown in the order of $\mathcal{O}(\sqrt{\log t/T_{h,i}(t)} \cdot \log t)$. This concentration is given by the boundedness (instead of sub-exponentiality) of clipped cycles, where the bounds grow logarithmically, which accounts for the extra $\log t$. □

Proof (*Proof Sketch of Theorem 1*)**.** Denote $\mathcal{T}^{(z)}$ as the collection of nodes in $\mathcal{T}$ with depth *greater than or equal to z*. Let $\mathcal{T}^{(z)} = \mathcal{T}_1 \cup \mathcal{T}_2$ where $\mathcal{T}_1 = \mathcal{D}(z, i_z^*)$ and $\mathcal{T}_2 = \mathcal{T}^{(z)} \setminus \mathcal{T}_1$.

By the conditional independence of cycle lengths/rewards (over cycles) and Lorden's inequality [31], one can show that the cumulative rewards $\mathsf{Rew}_{\pi(\boldsymbol{w}^*)}[\tau] = f^*\tau + \mathcal{O}(1)$. Hence,

$$\begin{aligned}
\mathsf{Reg}_{\pi}[\tau] &\leq f^*\tau - \mathbb{E}[\sum_{n=1}^{N_\pi[\tau]} \hat{U}_n] + \mathcal{O}(1) \\
&\leq \mathbb{E}[\sum_{n=1}^{N_\pi[\tau]} f^*\hat{C}_n - \hat{U}_n] + \mathcal{O}(\log\tau) \\
&\leq \sum_{j=1}^{2} \underbrace{\mathbb{E}[\sum_{n=1}^{N_\pi[\tau]} (f^*\hat{C}_n - \hat{U}_n)\mathbb{1}_{\{(H_n,I_n)\in\mathcal{T}_j\}}]}_{\mathsf{Reg}_{\pi,j}[\tau]} + \mathcal{O}(\log\tau).
\end{aligned}$$

We can prove that for $\mathcal{T}_1$,

$$\mathsf{Reg}_{\pi,1}[\tau] \leq \mu_{\max}\mathbb{E}[\sum_{n=1}^{\tau}\mathbb{1}_{\{(H_n,I_n)\in\mathcal{T}_1\}}(f^* - f(\boldsymbol{W}_n^\pi))] + \mathcal{O}(\log\tau).$$

Applying Lemma 1 and techniques used in the original HOO analysis [4], we show that

$$\mathsf{Reg}_{\pi,1}[\tau] = \mathsf{C}^{\mathsf{Reg}}\,(\mathsf{C}(\nu, \rho))^{\frac{1}{d(\nu,\rho)+2}}\,\tau^{\frac{d(\nu,\rho)+1}{d(\nu,\rho)+2}}(\log\tau)^{\frac{1}{d(\nu,\rho)+2}}$$

for a universal constant $\mathsf{C}^{\mathsf{Reg}}$. This involves further partitioning $\mathcal{T}_1$ in terms of whether or not a node is "near-optimal" with respect to its depth and smoothness parameters (ν, ρ), and handling subsets of $\mathcal{T}_1$ respectively.

For $\mathcal{T}_2$, we have that

$$\mathsf{Reg}_{\pi,2}[\tau] \leq l_\tau\mathbb{E}[\sum_{n=1}^{N_\pi[\tau]} \mathbb{1}_{\{(H_n,I_n)\in\mathcal{T}_2\}}]f^* + \mathcal{O}(\log n).$$

Note that any cycle before time τ is bounded by l_τ. The expected number of visits to $\mathcal{T}_2$ is at the order of $\mathcal{O}(\log^3 n)$ as a result of Assumption 4(a), implying that the number of "near-optimal" nodes is finite, and Lemma 2. □

Full proofs of the lemmas and the main theorem can be found in Appendix A.

4. Performance evaluation

In this section, we evaluate CHOOC from two experimental perspectives: (1) how well the algorithm converges (i.e., correctness and speed) in different settings, and (2) a series of useful scenarios the system might benefit from the CHOOC framework.

Basic Model Settings for Simulation: For experiments introduced in this section, we first model a simplified cellular wireless network to emulate stochastic channel and arrival processes. Suppose the CHOOC AHS is deployed at a Base Station (BS), located at the center of a circular cell of radius 250 m, which serves 10–20 active users. We assume the total bandwidth (BW) is 10 MHz which can be subdivided into 200 resource blocks. Each time slot lasts 0.5 ms.

For the arrival process of each user, we assume $A_i[t]$ has a *binomial* distribution. Each packet has a fixed size 5 kb. The *signal-to-interference ratio (SIR)* of user i at time t is modeled as $\mathtt{SIR}_i[t] = P_b g_i[t]/I_i[t]$, where $P_b, g_i[t], I_i[t]$ denote the

Table 2
Service rate vs User distance in our simulation system.

Distance to BS (m)	50	100	150	200	250
Mean Rate (packets/slot)	9.16	5.34	3.28	2.04	1.28

transmit power of the BS, channel gain, and interference level respectively. We set $P_b = 47$ dBm, and consider the channel gain as a combined effect of path loss, Rayleigh fast fading and an antenna gain of 17 dBi. Denote the user distance to BS by $\texttt{dist}_i$, the path loss is then modeled as $39.1\log_{10}(\texttt{dist}_i) + 13.5 + 20\log_{10}(f_c)$ where $f_c = 2.0$ GHz.[9] For simplicity, we assume the interference level is homogeneous to all users in the cell at −56 dBm. The service rate of user i is given by $S_i[t] = \texttt{BW} \times \log_2(1 + 10^{0.1(\texttt{SIR}_i[t]-\texttt{L})})$bps where $\texttt{L} = 3$ dB describes a gap relative to the ideal Shannon capacity. In the following simulations, we adjust the user distance to the BS to vary the channel distribution seen by users. Table 2 exhibits the mapping of a user's distance to mean service rates.

Suppose each transmitted packet is associated a reward, which is determined by the user type. Denote $\texttt{delay}$ as the packet delay in slots for any non-backlogged users' packet. We consider users of the following types:

(a) Mean-Delay Type (MD): The reward of each packet is given by $(1 - \texttt{delay} * 0.1)^+$.

(b) Deadline-(t) Type (DDL(t)): The packet reward equals the value of $\mathbb{1}_{\{\texttt{delay}<t\}}$, i.e., each packet has a deadline t.

(c) Backlogged Type (BL): The reward for each transmitted packet of an infinitely-backlogged queue is 1.

The cycle reward is defined as the sum of packet rewards of all users within a cycle, which is further normalized such that the cycle length/reward variables are of the same order. We set the cycle reward to be 0 if a cycle is clipped in order to penalize unstable systems. We use GPS as the slice-level scheduler throughout the simulation.

4.1. Convergence behavior of CHOOC

We explore the convergence behavior of CHOOC in our first experiments. We have observed that CHOOC always locates the *neighborhood of the best weight* within a reasonable time (note that one can only expect the algorithm to reach a certain level of optimality in limited time), accompanied with logarithmically-growing regret, which validates the main theorem of the last section. Below we consider convergence in the context of a 2-slice and a 3-slice setting.

4.1.1. A 2-slice system

We start with a simple 2-slice system, where each slice is associated with 6 users whose distances to the BS equal $50, 80, \ldots, 200$ m. Let $\mathcal{U}_1$ be MD users while $\mathcal{U}_2$ be DDL(7) users. The arrival process $A_i[t]$ is identical for all users, and is $\texttt{Binomial}(3, 0.1)$, i.e., $\lambda_i = 0.3$. The Log-Rule [33] is implemented as the flow-level scheduler for each slice. Let $\mathcal{W} = \{\boldsymbol{w} : |\boldsymbol{w}|_1 = 1, \boldsymbol{w} \succeq \boldsymbol{0}\}$.

To implement CHOOC, we set $\alpha = 4, \xi^2 = 1, \nu = 0.1, \rho = 0.5, \beta = 300, \kappa = 50, \mu_{\min} = 10, r_{\max} = 1$ and $z = 1$. Note that these parameter choices may be overly aggressive (to favor exploitation over exploration) to the extent that Assumption 3 may not hold. However, this is a common practice in UCB-type algorithms in order to see good convergence rate.

The top-left panel in Fig. 1 shows the (Monte Carlo-simulated) reward rate $f(\boldsymbol{w})$ and the number of clippings after simulating each of $\{\mathsf{HS}(\boldsymbol{w}) : w_1 = 0.05, \ldots, 0.95\}$ for 10k cycles under our proposed clipping rule $l_n = \beta + \kappa \log n$. The optimum is roughly $w_1 = 0.42$. We then run CHOOC for 10k cycles. In the bottom panels of Fig. 1, we show how the explored tree $\mathcal{T}_{\text{exp}}$ evolves from cycle index $n = 2$k to 10k, where each dot in the scatter plots represents a weight selection at the corresponding depth. As expected, the tree grows deeper around the optimum, implying that CHOOC is focusing on exploring near-optimal weights.

Next, we repeatedly simulate the same setting for 20 times and plot the median "near-optimal selection ratio" over cycles (exhibited at the top-right of Fig. 1), where the ratio at cycle index n is defined as the fraction of w_1 selections lying in $(w_1^* - \delta, w_1^* + \delta)$ for the first n cycles. We set $\delta = 0.05$ or 0.075. The AHS consistently finds the optimum's neighborhood. (Note that since f is relatively flat around $\boldsymbol{w}^*$, it is hard to concentrate the selections exactly at $\boldsymbol{w}^*$.) Convergence is further verified by the time-vs-regret plot in Fig. 1 (Top-Right), which shows a logarithmic growth. Recall that 1 s is equivalent to 2000 time slots.

Remark 6 (*Hyper-Parameter Choices*)**.** The convergence behavior is faster but more error-prone, if the parameters are set more "aggressively" to favor exploitation. For example, setting relatively-small μ and ρ (to accelerate contraction of the exploration bonus) has the benefit of "faster pruning", i.e., dropping out "obviously bad" regions more quickly, but risks ending up with sub-optimal solutions. In practice, one can use Assumption 3 as a (conservative) baseline when picking parameters and then adjust them properly.

[9] The SIR-related parameters stated above follow [32].

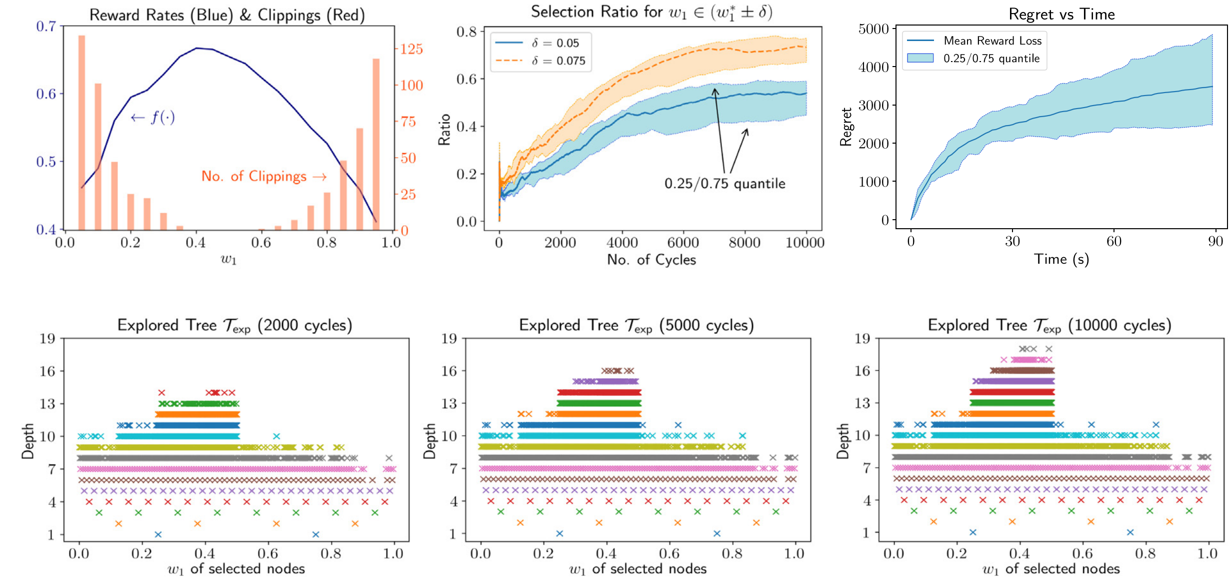


Fig. 1. Simulation results on experiments in Section 4.1. *(Top-Left)* Reward rate function $f(\boldsymbol{w})$ (left axis) and the number of clippings over 10000 cycles for the set of schedulers $\{\mathsf{HS}(\boldsymbol{w}) : w_1 = 0.05, \ldots, 0.95\}$ (right axis). *(Top-Middle)* Selection ratio for $w_1 \in (w_1^* \pm \delta)$ over the number of cycles: median, 0.25/0.75 quantile are shown after simulating CHOOC 20 times. *(Top-Right)* Total regret over time (20 simulation runs). *(Bottom)* Explored tree $\mathcal{T}_{\text{exp}}$ after running CHOOC for 2k, 5k and 10k cycles respectively. Each dot in the scatter plots represents a weight selection at the corresponding depth of the tree.

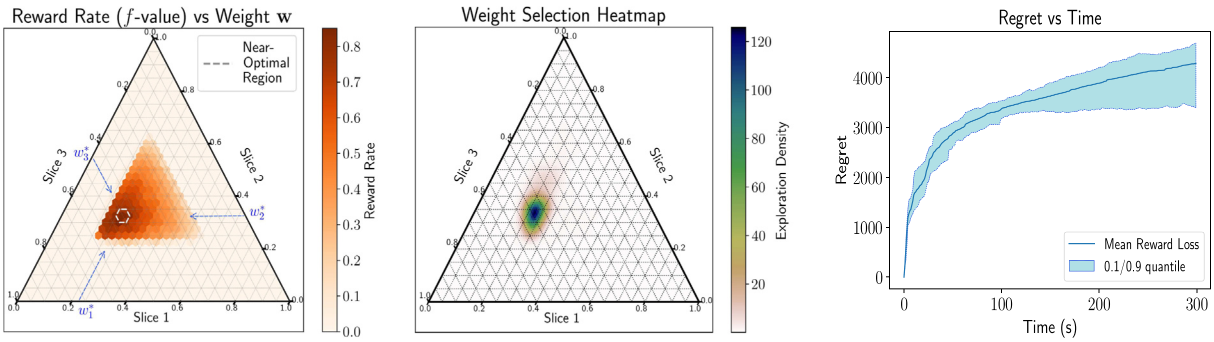


Fig. 2. Simulation results on experiments in Section 4.1.2. (*Left*) Reward rate f. *(Middle)* Heatmap for CHOOC's weight selections (6k cycles). *(Right)* Regret vs Time (averaging over 20 simulation runs).

Remark 7 (*Convergence in a High-Load System*). In the above simulation, we simulated a moderately-loaded system (the load was 3.6 packets/slot vs a mean service rate ~4.8) and the cycles were short (~10 ms). With higher loads, the set of system-stabilizing weights $\mathcal{W}(\boldsymbol{\lambda})$ contracts while the cycle time grows. The former effect helps the algorithm converge faster, since unstable weights induce much lower rewards (either intrinsically or penalized by clipping) and get eliminated sooner **in terms of the number of cycles used**. The latter effect delays the exploration of new weights, however, in practice it is possible to weaken the definition of cycles to further shorten the convergence time. For instance, one can require a cycle to terminate at time t such that $t = \arg\min_{t'}\{t' \geq t_0 + \tau_0 : \mathbf{Q}[t'] \in \mathcal{B}\}$, where t_0 is the cycle start time, τ_0 is a constant and $\mathcal{B}$ is a pre-specified bounded set in the queueing space. This means we only require the queues to return to a bounded region (rather than to the idle state) to finish a cycle, and an ideally-small $\mathcal{B}$ with a proper τ_0 (both serving as hyper-parameters) ensures that rewards are only *weakly correlated* across cycles. Later in Section 4.2.1, we will conduct an experiment using this relaxation and exhibit its performance.

4.1.2. A 3-slice system

In this part, we set up a 3-slice system to test the convergence of CHOOC in a more general weight space. Suppose each slice contains a mix of MD, DDL and BL type users (5 per slice) but the distributions are non-identical. The BS-to-user distance ranges from 50 to 200 m and the arrival rate is 0.25 packets/slot per user. We set $\mathcal{W} = \{\boldsymbol{w} : |\boldsymbol{w}|_1 = 1, w_i \geq 0.15, \forall i\}$.

Table 3
Simulation results on experiments in Section 4.2.1. For each setting, we list the f values for 4 weights: CHOOC-estimated weights for S1, S2, S3, and $\boldsymbol{w}_0 = [0.25, 0.25, 0.25, 0.25]$ (naive partition).

Grouping strategy	CHOOC-estimated weight $\tilde{\boldsymbol{w}}^*$	f^*	Reward rates $f(\cdot)$			
			$\tilde{\boldsymbol{w}}^*$(S1)	$\tilde{\boldsymbol{w}}^*$(S2)	$\tilde{\boldsymbol{w}}^*$(S3)	$\boldsymbol{w}_0$
S1	[0.33, 0.27, 0.28, 0.12]	*0.772*	**0.771**	0.752	0.768	0.755
S2	[0.18, 0.28, 0.29, 0.25]	*0.686*	0.635	**0.686**	0.680	0.664
S3	[0.20, 0.33, 0.28, 0.19]	*0.689*	0.650	0.659	**0.687**	0.661

The simulation-estimated function f is exhibited in Fig. 2 (Left). The optimal weight is close to [0.22, 0.33, 0.45], and the peripheral area with 0 reward rate indicates unstable weights. We run CHOOC (following the hyper-parameters in Section 4.1.1) for 6k cycles and draw a heatmap for its weight choices. As is shown in Fig. 2 (Middle), CHOOC faithfully concentrates on the optimal region, and those several layers around the optimum reflect the tree-based exploration process.

The regret over time is plotted as in Fig. 2 (Right). As a side note, there is no direct relation between the number of slices and CHOOC's convergence rate. The hardness of learning is determined explicitly by smoothness of f, which is measured by the near-optimality dimension $d(\mu, \rho)$ and its corresponding constant $\mathtt{C}(\mu, \rho)$. Roughly speaking, however, a higher dimension of $\mathcal{W}$ tends to induce a larger $\mathtt{C}(\mu, \rho)$, provided that the "curvature" of the function near its optimum (i.e., $d(\mu, \rho)$) is the same. This contributes to a larger coefficient term in the regret (see the main theorem).

4.2. Further motivating application scenarios

In this section, we showcase several scenarios where the best weight is hard to pre-determine due to the heterogeneity of user traffic, reward types or slicing structures, but all of which can be easily be optimized by our CHOOC framework. These experiments further exhibit the potential of CHOOC for learning the best weights for hierarchical scheduling in complicated environments.

4.2.1. Different grouping strategies

Sometimes users are grouped into slices based on similar traits, but the criteria for grouping may vary. In the following, we set up a 4-slice, 16-user system and apply CHOOC over varied grouping strategies of heterogeneous users into slice partitions.

There are 4 types of users: DDL(2), DDL(3), DDL(4), DDL(5). Associated with each type there are 4 users whose distances to the BS are 50, 80, 110, 140 m respectively. The arrival process is homogeneous for each user i and $\lambda_i = 0.32$. Since this system contains more users with higher loads, we adopt the weakened cycle notion proposed in Remark 7 to ensure faster convergence, and set $\mathcal{B} = \{\boldsymbol{Q} : \max_i Q_i \le 2\}$ and $\tau_0 = 40$. (As a result, cycles under the best weights take ~ 50 ms.)

When setting up the hierarchical scheduler, we consider three grouping strategies for slice partitioning: (S1) users with the same distance to BS are grouped together; (S2) users with the same deadline requirement are grouped together;We set $\mathcal{W} = \{\boldsymbol{w} : w_i \ge 0.1, \forall i\}$, providing a level of isolation among slices.

We first estimate the best reward rate f^* of each scenario by grid searching the simplex $\mathcal{W}$ and Monte-Carlo simulations. Then we run CHOOC in the three settings respectively for 720k time slots (equivalent to 360 s), and we define the *CHOOC-estimated weight* $\tilde{\boldsymbol{w}}^*$ as the mean of weight parameters for last 2k cycles.[10] Finally, we compute $f(\tilde{\boldsymbol{w}}^*)$ by numerical simulation to validate the correctness of CHOOC. The results of f^*, $\tilde{\boldsymbol{w}}^*$ and $f(\tilde{\boldsymbol{w}}^*)$ are summarized in Table 3.

As one can expect, the best weight (or say the set of near-optimal weights), despite being easily learned by the CHOOC algorithm, is highly unpredictable due to the complicated trade-offs among users with different service rates and requirements. For example in S1, the optimal weight allocation indeed sacrifices users with the worst service rates (Slice 4) so as to achieve a higher rate of sum rewards. To further validate this observation, for each setting, we compare f values of all three CHOOC-estimated weights as well as a naive choice $\boldsymbol{w}_0 = [0.25, 0.25, 0.25, 0.25]$. It turns out the naive partition is far from ideal and the best weights cannot be transferred across settings.

For each setting, we plot the regret over time (averaged over 20 CHOOC simulation runs) as shown in Fig. 3. It is worth noting that S1 has a much better regret mainly due to the fact that the stable set $\mathcal{W}(\boldsymbol{\lambda})$ for S1 is larger (i.e., lower chance to explore low-reward unstable weights inducing long cycles).

4.2.2. Imbalanced number of users across slices

In the next set of simulation, we test the performance of CHOOC under variations in the number of users in each slice. We set up an 8-user, 2 slice system. All users are of type MD with arrival rates being 0.58 packets/slot and the distances

[10] When approximating the f function of this 4-slice system, we observe that for each setting there exists a subset of weights that all correspond to "near-optimal" reward rates, and it is computationally-hard to estimate the *exact* best weight $\boldsymbol{w}^*$ by numerical simulation due to stochastic error and the complexity of f (compared to the previous 2-slice system where f is a simple concave 1D function). For the same reason, the CHOOC-estimated weight indeed varies over simulation runs (within a near-optimal region), and here in Table 3 we only display one representation.

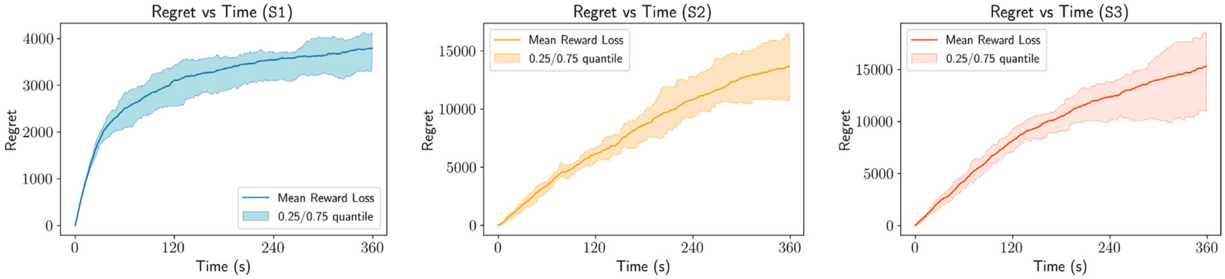


Fig. 3. Regret plots on experiments in Section 4.2.1.

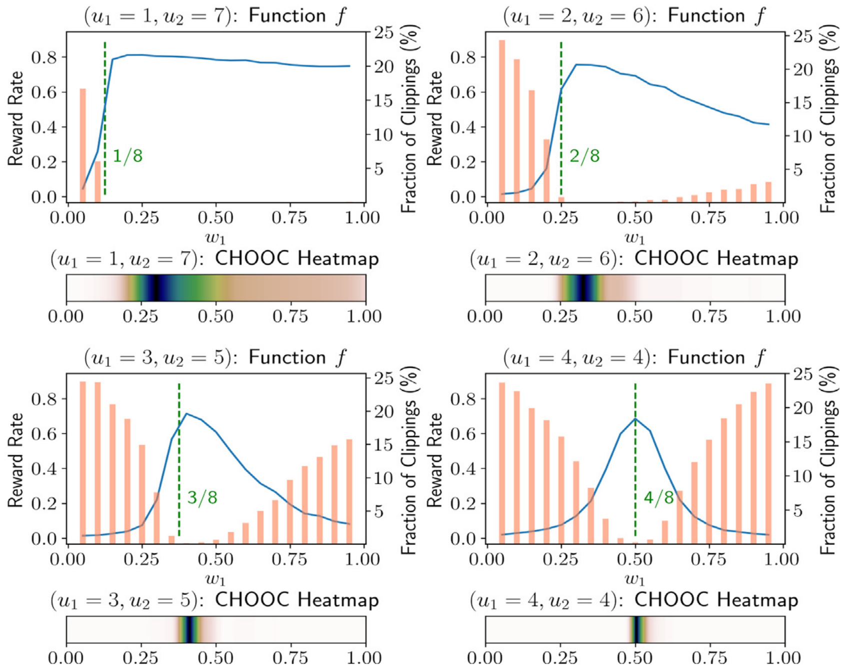


Fig. 4. Simulation results on experiments in Section 4.2.2. For each scenario, the reward rate function and the corresponding weight density heatmap selected by CHOOC are displayed. Note that the simple heuristic $|\mathcal{U}_1|/|\mathcal{U}|$ (dashed line) does not always match the optimal weight.

to BS equaling 120 m. Both slices implement a Log-Rule scheduler. We assign the number of users to the two slices as being $(1, 7), (2, 6), (3, 5), (4, 4)$. In Fig. 4, we plot the simulation-approximated f values for each scenario. We observe that, except in the last scenario, the optimal weight w_1^* is not $|\mathcal{U}_1|/|\mathcal{U}|$ (proportional to the number of users) as one might suggest as a reasonable heuristic. This is due to the joint effect of the asymmetry in the number of users per slice and opportunistic scheduling within slices.[11]

For each case, we run CHOOC for 10k cycles and the convergence is shown by a density heatmap (see Fig. 4). As one can imagine, the selected weights are less concentrated around $\boldsymbol{w}^*$ for the first case, since f is flatter.

4.2.3. Tradeoffs between backlogged and non-backlogged users

In this set of simulations, we set up a scenario exhibiting the ability of CHOOC to realize tradeoffs among slices of backlogged and non-backlogged users. Due to page limit, we include these results in Appendix D.1.

4.2.4. Different scheduler choices in CHOOC framework.

We investigate the performance of CHOOC under various scheduler implementations. These results are included in Appendix D.2.

[11] Note that a GPS scheduler will re-assign unused resources by $\mathcal{U}_1$ to $\mathcal{U}_2$. Thus, $w_1 = 1$ does not mean that $\mathcal{U}_2$ receives no resources; instead, it implies $\mathcal{U}_1$ is given the full priority. This explains why in the top left panel in Fig. 4, f is almost flat for $w_1 > 0.2$ as $\mathcal{U}_1$ already receives sufficient resources.

5. Conclusion

We study a hierarchical scheduling model for network slicing where the optimal resource allocation among slices, parameterized by a weight vector, is to be determined. To address the complexities of the problem (diversity in user traffic or service requirements), we formulate this through a MAB blackbox optimization framework. We propose the CHOOC algorithm (an adaptive hierarchical scheduler) that builds on the classical HOO with algorithmic/theoretic modifications to account for cycles with clippings, and thus is applicable in a queueing-based slicing/scheduling wireless system. Our simulations show that our algorithm has the ability to find the best weight in various scenarios. Although a GPS-based hierarchical scheduler is considered in this paper, we note that the CHOOC algorithm can be generalized to other parameterized scheduling model with cycles. It is thus of interest to explore its benefits in other related settings.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.peva.2021.102237.

References

[1] O. Sallent, J. Perez-Romero, R. Ferrus, R. Agusti, On radio access network slicing from a radio resource management perspective, IEEE Wirel. Commun. 24 (5) (2017) 166–174.
[2] R. Srikant, L. Ying, Communication Networks: An Optimization, Control, and Stochastic Networks Perspective, Cambridge University Press, 2013.
[3] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, R. Vijayakumar, P. Whiting, Scheduling in a queuing system with asynchronously varying service rates, Probab. Engrg. Inform. Sci. 18 (2) (2004) 191–217.
[4] S. Bubeck, R. Munos, G. Stoltz, C. Szepesvári, X-armed bandits, J. Mach. Learn. Res. 12 (5) (2011).
[5] X. Foukas, G. Patounas, A. Elmokashfi, M.K. Marina, Network slicing in 5G: Survey and challenges, IEEE Commun. Mag. 55 (5) (2017) 94–100.
[6] X. Costa-Pérez, J. Swetina, T. Guo, R. Mahindra, S. Rangarajan, Radio access network virtualization for future mobile carrier networks, IEEE Commun. Mag. 51 (7) (2013) 27–35.
[7] T. Guo, A. Suárez, Enabling 5G RAN slicing with EDF slice scheduling, IEEE Trans. Veh. Technol. 68 (3) (2019) 2865–2877.
[8] J. Zheng, P. Caballero, G. De Veciana, S.J. Baek, A. Banchs, Statistical multiplexing and traffic shaping games for network slicing, IEEE/ACM Trans. Netw. 26 (6) (2018) 2528–2541.
[9] V. Sciancalepore, X. Costa-Perez, A. Banchs, Rl-NSB: Reinforcement learning-based 5G network slice broker, IEEE/ACM Trans. Netw. 27 (4) (2019) 1543–1557.
[10] F. Wei, G. Feng, Y. Sun, Y. Wang, S. Qin, Y.-C. Liang, Network slice reconfiguration by exploiting deep reinforcement learning with large action space, IEEE Trans. Netw. Serv. Manag. 17 (4) (2020) 2197–2211.
[11] R. Li, C. Wang, Z. Zhao, R. Guo, H. Zhang, The LSTM-based advantage actor-critic learning for resource management in network slicing with user mobility, IEEE Commun. Lett. 24 (9) (2020) 2005–2009.
[12] X. Shen, J. Gao, W. Wu, K. Lyu, M. Li, W. Zhuang, X. Li, J. Rao, Ai-assisted network-slicing based next-generation wireless networks, IEEE Open J. Veh. Technol. 1 (2020) 45–66.
[13] S.K. Singh, M.M. Salim, J. Cha, Y. Pan, J.H. Park, Machine learning-based network sub-slicing framework in a sustainable 5g environment, Sustainability 12 (15) (2020) 6250.
[14] S. Bubeck, N. Cesa-Bianchi, et al., Regret analysis of stochastic and nonstochastic multi-armed bandit problems, Found. Trends® Mach. Learn. 5 (1) (2012) 1–122.
[15] T. Lattimore, C. Szepesvári, Bandit Algorithms, Cambridge University Press, 2020.
[16] R. Kleinberg, A. Slivkins, E. Upfal, Multi-armed bandits in metric spaces, in: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, 2008, pp. 681–690.
[17] R. Munos, Optimistic optimization of a deterministic function without the knowledge of its smoothness, in: Advances in Neural Information Processing Systems, 2011, pp. 783–791.
[18] M. Valko, A. Carpentier, R. Munos, Stochastic simultaneous optimistic optimization, in: International Conference on Machine Learning, 2013, pp. 19–27.
[19] J.-B. Grill, M. Valko, R. Munos, Black-box optimization of noisy functions with unknown smoothness, in: Advances in Neural Information Processing Systems, 2015, pp. 667–675.
[20] R. Sen, K. Kandasamy, S. Shakkottai, Noisy blackbox optimization using multi-fidelity queries: A tree search approach, in: The 22nd International Conference on Artificial Intelligence and Statistics, 2019, pp. 2096–2105.
[21] P.L. Bartlett, V. Gabillon, M. Valko, A simple parameter-free and adaptive approach to optimization under a minimal local smoothness assumption, in: Algorithmic Learning Theory, 2019, pp. 184–206.
[22] C. Fiegel, V. Gabillon, M. Valko, Adaptive multi-fidelity optimization with fast learning rates, in: International Conference on Artificial Intelligence and Statistics, 2020, pp. 3493–3502.
[23] S.H.A. Ahmad, M. Liu, Multi-channel opportunistic access: A case of restless bandits with multiple plays, in: 2009 47th Annual Allerton Conference on Communication, Control, and Computing (Allerton), IEEE, 2009, pp. 1361–1368.
[24] Y. Gai, B. Krishnamachari, R. Jain, Learning multiuser channel allocations in cognitive radio networks: A combinatorial multi-armed bandit formulation, in: 2010 IEEE Symposium on New Frontiers in Dynamic Spectrum, DySPAN, IEEE, 2010, pp. 1–9.
[25] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, Z. Niu, Learning-based task offloading for vehicular cloud computing systems, in: 2018 IEEE International Conference on Communications, ICC, IEEE, 2018, pp. 1–7.
[26] I. Tariq, R. Sen, G. de Veciana, S. Shakkottai, Online channel-state clustering and multiuser capacity learning for wireless scheduling, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 136–144.

[27] S. Cayci, A. Eryilmaz, R. Srikant, Learning to control renewal processes with bandit feedback, Proc. ACM Meas. Anal. Comput. Syst. 3 (2) (2019) 43.

[28] J. Song, G. de Veciana, S. Shakkottai, Meta-scheduling for the wireless downlink through learning with bandit feedback, in: Proc. IEEE WIOPT Workshop on Machine Learning in Wireless Communications, WMLC, 2020, pp. 1–7, Invited paper.

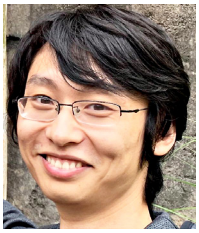
[29] B. Hajek, Hitting-time and occupation-time bounds implied by drift analysis with applications, Adv. Appl. Probab. 14 (3) (1982) 502–525.

[30] W. Ding, T. Qin, X.-D. Zhang, T.-Y. Liu, Multi-armed bandit with budget constraint and variable costs, in: Twenty-Seventh AAAI Conference on Artificial Intelligence, 2013.

[31] G. Lorden, On excess over the boundary, Ann. Math. Stat. (1970) 520–527.

[32] M. Series, Guidelines for evaluation of radio interface technologies for IMT-advanced, Rep. ITU 638 (2009).

[33] B. Sadiq, S.J. Baek, G. De Veciana, Delay-optimal opportunistic scheduling and approximations: The log rule, IEEE/ACM Trans. Netw. 19 (2) (2011) 405–418.



Jianhan Song received the B.S. degree in information engineering from The Chinese University of Hong Kong in 2017. He is currently pursuing the joint M.S. and Ph.D. degree with the Department of Electrical and Computer Engineering, The University of Texas at Austin. He also joins the Wireless Networking and Communications Group (WNCG), The University of Texas at Austin, as a Graduate Research Assistant. His research interests include queuing systems, wireless networking, and reinforcement learning.

Gustavo de Veciana received his B.S., M.S, and Ph.D. in electrical engineering from the University of California at Berkeley in 1987, 1990, and 1993 respectively. He is currently a Professor and Associate Chair of the Department of Electrical and Computer Engineering and recipient of the Cockrell Family Regents Chair in Engineering. He served as the Director and Associate Director of the Wireless Networking and Communications Group (WNCG) at the University of Texas at Austin, from 2003-2007. His research focuses on the design, analysis and control networks, information theory and applied probability. Current interests include: measurement, modeling and performance evaluation; wireless and sensor networks; architectures and algorithms to design reliable computing and network systems. Dr. de Veciana has been an editor for the IEEE/ACM Transactions on Networking. He was the recipient of a National Science Foundation CAREER Award 1996, and co-recipient of 6 best paper awards, and recipient of the IEEE Bennet Prize in 2021. In 2009 he was designated IEEE Fellow for his contributions to the analysis and design of communication networks. He currently serves on the board of trustees of IMDEA Networks Madrid.

Sanjay Shakkottai received his Ph.D. from the ECE Department at the University of Illinois at Urbana-Champaign in 2002. He is with The University of Texas at Austin, where he is a Professor in the Department of Electrical and Computer Engineering, and holds the Cockrell Family Chair in Engineering #15. He received the NSF CAREER award in 2004 and was elected as an IEEE Fellow in 2014. He was a co-recipient of the IEEE Communications Society William R. Bennett Prize in 2021. His research interests lie at the intersection of algorithms for resource allocation, statistical learning and networks, with applications to wireless communication networks and online platforms.