Joint Learning and Control in Stochastic Queueing Networks with Unknown Utilities

XINZHE FU, LIDS, Massachusetts Institute of Technology, USA
EYTAN MODIANO, LIDS, Massachusetts Institute of Technology, USA

We study the optimal control problem in stochastic queueing networks with a set of job dispatchers connected to a set of parallel servers with queues. Jobs arrive at the dispatchers and get routed to the servers following some routing policy. The arrival processes of jobs and the service processes of servers are stochastic with unknown arrival rates and service rates. Upon the completion of each job from dispatcher u_n at server s_m, a random utility whose mean is unknown is obtained. We seek to design a control policy that makes routing decisions at the dispatchers and scheduling decisions at the servers to maximize the total utility obtained by the end of a finite time horizon T. The performance of policies is measured by regret, which is defined as the difference in total expected utility with respect to the optimal dynamic policy that has access to arrival rates, service rates and underlying utilities.

We first show that the expected utility of the optimal dynamic policy is upper bounded by T times the solution to a static linear program, where the optimization variables correspond to rates of jobs from dispatchers to servers and the feasibility region is parameterized by arrival rates and service rates. We next propose a policy for the optimal control problem that is an integration of a learning algorithm and a control policy. The learning algorithm seeks to learn the optimal extreme point solution to the static linear program based on the information available in the optimal control problem. The control policy, a mixture of priority-based and Joint-the-Shortest-Queue routing at the dispatchers and priority-based scheduling at the servers, makes decisions based on the graphical structure induced by the extreme point solutions provided by the learning algorithm. We prove that our policy achieves logarithmic regret whereas application of existing techniques to the optimal control problem would lead to $\Omega(\sqrt{T})$-regret. The theoretical analysis is further complemented with simulations to evaluate the empirical performance of our policy.

CCS Concepts: • **Networks** → **Network algorithms**; • **Theory of computation** → **Theory and algorithms for application domains**.

Additional Key Words and Phrases: Stochastic Queueing Networks; Learning For Network Control

ACM Reference Format:
Xinzhe Fu and Eytan Modiano. 2022. Joint Learning and Control in Stochastic Queueing Networks with Unknown Utilities. *Proc. ACM Meas. Anal. Comput. Syst.* 6, 3, Article 58 (December 2022), 32 pages. https://doi.org/10.1145/3570619

1 INTRODUCTION

Consider a bipartite queueing network with a set of job dispatchers connected to a set of servers with queues. Jobs arriving at the dispatchers get routed to the queues for service, while a certain utility is obtained for each job completed. Such bipartite queueing networks have been widely adopted to model networked systems such as inter-connected switches [1], cloud platforms [2] and server farms [3, 4]. We study the optimal control problem in such bipartite queueing networks.

Authors' addresses: Xinzhe Fu, LIDS, Massachusetts Institute of Technology, USA; Eytan Modiano, LIDS, Massachusetts Institute of Technology, USA.

2476-1249/2022/12 – Art 58. https://doi.org/10.1145/3570619

Jobs of unit size arrive at the dispatchers and are sent to a server following some routing policy. Each server has a queue that buffers the incoming jobs and serves the jobs in the queue following some scheduling policy. The job arrival process at each dispatcher u_n is a stochastic process with unknown arrival rate λ_n and the service process of each server s_m is stochastic with unknown service rate μ_m (see Figure 1 for an illustration). Upon the completion of a job from dispatcher u_n at server s_m, a random utility $\hat{v}_{nm}$ is obtained with an unknown mean $\mathbb{E}[\hat{v}_{nm}] = v_{nm}$. The utility of each job is independent. The goal is to design a control policy that minimizes regret, which is defined as the difference in the total expected utility obtained by the end of a finite time-horizon T with respect to the optimal dynamic policy that has the knowledge of arrival rates, service rates and underlying utilities. In this paper, we will propose a policy that achieves logarithmic regret, i.e., the regret of the policy grows (poly)logarithmically with the time horizon T.

First, we show that the expected utility of the optimal dynamic policy is upper bounded by T times the value of a linear program, which we will refer to as the static linear program of the optimal control problem. The static linear program can be interpreted as a fluid version of the optimal control problem, where the optimization variables correspond to rates of jobs from the dispatchers to the servers, the feasibility region is the set of rates that satisfy the resource constraints of the arrivals and the capacity constraints of the servers, and the objective function is the total utility corresponding to the rates (see Figure 2 for a concrete example). Note that both the objective function, which involves underlying utilities, and the feasibility region, which involves arrival rates and service rates, of the static linear program are unknown apriori. Thus, the optimal control problem requires a combination of learning and network control, i.e., it needs an algorithm that seeks to learn the solution to the static linear program based on observed information, and a control policy that makes routing and scheduling decisions based on the learned solution.

Optimization problems with unknown parameters have received considerable attention in the online decision making literature. Since the classical multi-armed bandit problem[11], there has been extensive effort dedicated to optimization problems with unknown objective functions of various form [5–9, 17]. Previous works focus on finding the minimizer of a linear [17] or convex [5, 9] objective function in a given feasibility region, where the objective function is unknown but feedback on function values can be observed. Subsequent works extend the consideration to problems with cumulative constraints over the time horizon [25, 26] or stochastic constraints that are unknown apriori but revealed sequentially over time [5, 27].

In contrast to the growing literature on problems with unknown parameters in pure optimization settings, the stochastic queueing dynamics in optimal control brings new challenges to both learning and network control. For the learning aspect, different from traditional problems in the literature where the objective function value is immediately observed upon decision on the optimization variables, in the optimal control problem utility observations experience queueing delay as they are only available after the jobs complete service at the servers. Therefore, the learning algorithm for solving the static linear program must be able to handle the feedback delay in addition to the unknown feasibility region in stochastic queueing networks.

From the aspect of network control, first, it is worth noting that even if the optimal solution to the static linear program was given exactly, designing a control policy with logarithmic regret is a non-trivial task. Indeed, for example, suppose we are given the optimal solution $r^*_{11} = 5, r^*_{12} = 2$ to the linear program of Figure 2, an immediate candidate is a static randomized routing policy that every time sends the incoming jobs to s_1 with probability $\frac{5}{7}$ and to s_2 with probability $\frac{2}{7}$ combined with an arbitrary scheduling policy at the servers. Such a policy will lead to the queue of s_1 being critically loaded and result in $\Omega(\sqrt{T})$-regret. In fact, we will formally show in Section 4 that any static policy will incur $\Omega(\sqrt{T})$-regret, which is strictly worse than the policy we propose that

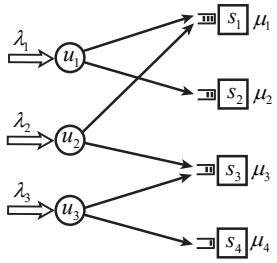

Fig. 1. An example of the bipartite queueing network model with three job dispatchers and four servers.

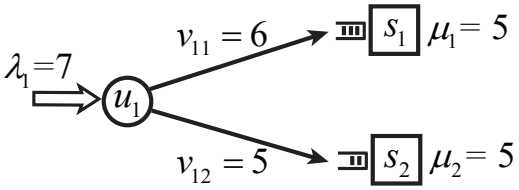

Fig. 2. A simple example with one dispatcher and two servers. In this example, the static linear program is to maximize $v_{11}r_{11} + v_{12}r_{12}$ subject to $r_{11} + r_{12} = \lambda_1, r_{11} \leq \mu_1, r_{12} \leq \mu_2$.

achieves logarithmic regret. What further elevates the challenge is that in the optimal control problem, as the parameters are unknown and we can only observe their stochastic realizations, it is impossible to obtain the exact optimal solution to the static linear program, e.g., no learning algorithm can obtain the exact values of r_{11}^*, r_{12}^* in a finite time horizon [17]. The solutions computed by the learning algorithm are inherently approximate, which precludes the routing policy from relying on the values of the solutions as the approximation error will accumulate to $\Omega(\sqrt{T})$-regret (See Section 4). Instead, the routing policy has to rely on some structure of the solutions that is robust to the approximation error.

Our main results are as follows. First, we show that each extreme point solution to the static linear program induces a spanning forest of the bipartite network. We thus propose a control policy that consists of a mixture of priority-based routing and Join-the-Shortest-Queue routing at the dispatchers and priority-based scheduling at the servers, which relies on the spanning forests of the solutions as input. Such spanning forest structure is robust against the error in the values of the solutions. Second, we propose a learning algorithm that can learn the optimal solution to the linear program and handle the unknown feasibility region in the optimal control problem. Finally, we integrate the learning algorithm and the control policy into a joint learning and control policy. We show that our policy achieves logarithmic regret for the optimal control problem, which is superior to application of existing techniques that will lead to $\Omega(\sqrt{T})$-regret. We also complement our theoretical analysis with empirical evaluation. Our results highlight the importance of co-design for the learning and the networking control aspects of the optimal control problem.

We note that the optimal control problem can be subsumed into the general framework of network utility maximization (with unknown utility functions) [8, 23, 24] or reinforcement learning [12–14]. However, the methods therein can also only achieve $\Omega(\sqrt{T})$-regret since they cannot leverage the structure of the optimal control problem. We refer the reader to Section 4 for further discussion.

The rest of the paper is organized as follows: we formally present the model and formulation of the optimal control problem in Section 2. In Section 3, we introduce several key preliminary results for the optimal control problem. In Section 4, we will give an overview of our main results and provide discussion on related results in the literature. We propose our control policy in Section 5, and the learning algorithm and the joint learning and control policy in Section 6. In Section 7, we conduct empirical evaluation on our policy. Finally, we conclude the paper in Section 8.

2 MODEL AND PROBLEM FORMULATION

Consider a bipartite network $\mathcal{G}(\mathcal{U}, \mathcal{S})$ that operates in discrete time with N job dispatchers $\{u_1, \ldots, u_N\}$ and M parallel servers $\{s_1, \ldots, s_M\}$ (see Figure 1). The job dispatchers are controlled by a single decision maker. Dispatcher u_n is connected to a set of servers $\mathcal{S}_n$. Server s_m is connected to a set of job dispatchers $\mathcal{U}_m$. We will also sometimes use n to represent a generic dispatcher and m to represent a generic server. At each time slot t, $a_n(t)$ unit-size jobs arrive at dispatcher u_n. The arrivals $a_n(t)$'s are independent random variables with unknown means (arrival rates) $\mathbb{E}[a_n(t)] = \lambda_n$. Each dispatcher sends its incoming jobs to servers to which it is connected. Each server has a buffer that stores incoming jobs. For server s_m, its offered service at time t is denoted by $c_m(t)$, with $c_m(t)$'s being independent random variables with unknown mean (service rate) $\mathbb{E}[c_m(t)] = \mu_m$. The offered service $c_m(t)$ is equal to the number of jobs that s_m can finish executing at time t. The arrival rates $\{\lambda_n\}$ and the service rates $\{\mu_m\}$ will be referred to as the network statistics. After a job j from dispatcher u_n finishes execution at server s_m, we obtain and observe a random utility $\hat{v}_{nm}(j)$ with an unknown mean $\mathbb{E}[\hat{v}_{nm}(j)] = v_{nm}$, where v_{nm} is the underlying utility associated with server s_m for dispatcher u_n. Each $\hat{v}_{nm}(j)$ is a 1-sub-Gaussian random variable[1] and $\hat{v}_{nm}(j)$'s for different jobs are independent. Note that the underlying utilities v_{nm}'s are unknown, and that we can only observe $\hat{v}_{nm}(j)$'s. We assume that the realized arrivals and offered service rates, i.e., $a_n(t)$'s, $c_m(t)$'s are all bounded by a constant C almost surely. Finally, let $a_{nm}(t)$ be the number of jobs from dispatcher u_n sent to server s_m at time t by the job dispatcher and $Q_m(t)$ be the queue length of server m at time t. The evolution of queue length can be written with the Lindley recursion: $Q_m(t+1) \coloneqq \left[Q_m(t) + \sum_{n \in \mathcal{U}_m} a_{nm}(t) - c_m(t)\right]^+$, where $[\cdot]^+ = \max\{\cdot, 0\}$.

Our goal is to design a control policy that makes routing decisions at the dispatchers(i.e. sending each incoming job to a server) and scheduling decisions at the servers (i.e. deciding which jobs to serve) such that the expected utility obtained by the end of the time horizon T is maximized. To make the problem concrete, we first define the expected utility of a generic policy π. Let C_{nm}^{π} be the total number of jobs from dispatcher u_n completed at server s_m by the end of the time horizon. Note that C_{nm}^{π} is a random variable. As the noise associated with the utility of each job is independent, the expected total utility obtained under π is $U(\pi) = \sum_{n=1}^{N} \sum_{m=1}^{M} v_{nm} \mathbb{E}[C_{nm}^{\pi}]$. Also, note that only the jobs that are completed by T contribute to the total utility while the jobs that are left in the queue at the end of the time horizon T do not count towards the total utility. Let Π be the set of all policies, including the ones that have knowledge of the underlying utilities $\{v_{nm}\}$, the network statistics, and the realizations of arrivals and services over the whole time horizon. The optimal dynamic policy π^* is the best policy in Π, i.e., $\pi^* = \arg\max_{\pi' \in \Pi} U(\pi')$. Note that the optimal dynamic policy is typically inadmissible in the optimal control problem as it can make decisions based on information that is not available in the problem setting. We define the regret of a policy π as $R(\pi) = U(\pi^*) - U(\pi)$, i.e., the gap between the expected utility of π and the optimal dynamic policy. In this paper, we pursue admissible policies that make decisions only based on observable information without prior knowledge of the network statistics or the underlying utilities with low regret. We will refer to the problem as the *Optimal Control Problem*.

3 PRELIMINARIES

In this section, we introduce several key preliminary results. We start by presenting a static linear program as fluid version of the optimal control problem. Based on the linear program, we first give an upper bound on the expected utility of the optimal dynamic policy, and then set up the

[1] A random variable X is 1-sub-Gaussian if $\mathbb{P}\{|X - \mathbb{E}[X]| \geq t\} \leq 2\exp(-\frac{t^2}{2})$, i.e., its tail is dominated by a Gaussian distribution with variance 1.

instance-dependent conditions on the network statistics that will be assumed throughout the paper. Finally, we establish several structural properties of the extreme points of the linear program.

3.1 The Static Linear Program

Consider the following linear program $\mathcal{P}$.

$$\mathcal{P}: \max_{\{r_{mn}\}} \sum_{n=1}^{N} \sum_{m \in \mathcal{S}_n} v_{nm} r_{nm} \tag{1}$$

$$\textbf{s.t.} \quad \sum_{m \in \mathcal{S}_n} r_{nm} = \lambda_n, \tag{2}$$

$$\sum_{n: s_m \in \mathcal{S}_n} r_{nm} \leq \mu_m. \tag{3}$$

$$r_{nm} \geq 0. \tag{4}$$

The linear program $\mathcal{P}$ can be interpreted as a fluid version of the optimal control problem, where r_{nm} represents the rate of jobs from dispatcher u_n completed at server s_m. Let $OPT(\mathcal{P})$ be the optimal value of $\mathcal{P}$. Using the same argument as in [23], we have the following proposition. Similar results that upper-bound the value of a stochastic dynamic program with its fluid counterpart have appeared in other works on online decision problems, e.g. [25, 26].

Proposition 1. *The expected utility of the optimal dynamic policy is upper bounded by $T \cdot OPT(\mathcal{P})$, i.e., $U(\pi^*) \leq T \cdot OPT(\mathcal{P})$.*

From Proposition 1, we see that if our policy can obtain an expected utility close to $T \cdot OPT(\mathcal{P})$, then it will achieve low regret. We can thus approach the optimal control problem through learning the solution to $\mathcal{P}$ while making routing decisions based on the learned solution.

3.2 Structure of the Extreme Points

We now briefly recall some standard definitions from the linear programming literature [29] that will be useful in the proceeding discussion. The definitions are most conveniently stated for a linear program in standard form: $\min_{\boldsymbol{x}} \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}$ s.t. $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$, with $\boldsymbol{A} \in \mathbb{R}^{m \times n}, m \leq n$, the feasibility region $\mathcal{D} = \{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$ is a polyhedron. Let $\boldsymbol{x}^*$ be an optimal solution and $\mathcal{E}$ be the set of all extreme points of the polyhedron $\mathcal{D}$, which is equivalently the set of all basic feasible solutions of the linear program. Without loss of generality, assuming A has m linearly-independent rows, each basic solution is characterized by a basis $\boldsymbol{B} = [A_{B(1)}, \ldots A_{B(m)}]$, where $A_{B(1)}, \ldots, A_{B(m)}$ are m linearly-independent columns of $\boldsymbol{A}$ with $B(1), \ldots, B(m)$ being the indices of the columns. For a generic vector $\boldsymbol{x}$, we write $\boldsymbol{x}_{\boldsymbol{B}}$ as the coordinates of $\boldsymbol{x}$ corresponding to indices $B(1), \ldots, B(m)$. A basis is feasible if and only if $\boldsymbol{B}^{-1}\boldsymbol{b} \geq \boldsymbol{0}$. Write $(\boldsymbol{B}^{-1}\boldsymbol{b})_i$ as the i-th component of $\boldsymbol{B}^{-1}\boldsymbol{b}$. A basis $\boldsymbol{B}$ is Δ-feasible if $\min_{i:(\boldsymbol{B}^{-1}\boldsymbol{b})_i > 0} (\boldsymbol{B}^{-1}\boldsymbol{b})_i \geq \Delta$, i.e., the minimum positive basis variable is greater than Δ. A basis $\boldsymbol{B}$ is Δ-infeasible if $\max_{i:(\boldsymbol{B}^{-1}\boldsymbol{b})_i < 0} (\boldsymbol{B}^{-1}\boldsymbol{b})_i \leq -\Delta$, i.e., the maximum negative basis variable is smaller than -Δ.

Each basic feasible solution is characterized by a feasible basis, i.e.,

$$\mathcal{E} = \{\boldsymbol{x} \mid x_{\boldsymbol{B}} = \boldsymbol{B}^{-1}\boldsymbol{b} \geq \boldsymbol{0}, \text{ for some } \boldsymbol{B} = [A_{B(1)}, \ldots A_{B(m)}], x_i = 0 \text{ for } i \notin \{B(1), \ldots, B(m)\}\}$$

If the feasibility region $\mathcal{D}$ is bounded, then there exists an optimal solution $\boldsymbol{x}^*$ that is a basic feasible solution or equivalently, an extreme point (i.e., $\boldsymbol{x}^* \in \mathcal{E}$). We will write $\boldsymbol{B}^*$ as the basis associated with $\boldsymbol{x}^*$. Following the standard definition in the linear programming literature [29], we say an extreme point $\boldsymbol{x}$ with basis B is non-degenerate if $\boldsymbol{x}_B > \boldsymbol{0}$ component-wise.

Back to the static linear program $\mathcal{P}$ of the optimal control problem, we can write $\mathcal{P}$ in standard form as the following.

$$\begin{aligned}
\mathcal{P}: \quad & \max_{\{r_{nm}, y_m\}} \sum_{n=1}^{N} \sum_{m \in \mathcal{S}_n} v_{nm} r_{nm} \\
\textbf{s.t.} \quad & \sum_{m \in \mathcal{S}_n} r_{nm} = \lambda_n, \\
& \sum_{n: s_m \in \mathcal{S}_n} r_{nm} + y_m = \mu_m. \\
& r_{nm}, y_m \geq 0.
\end{aligned}$$

Here, the matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$ are formed by the constraints $\sum_{m \in \mathcal{S}_{u_n}} r_{nm} = \lambda_n$ and $\sum_{n: s_m \in \mathcal{S}_{u_n}} r_{nm} + y_m = \mu_m$ and the optimization vector $\boldsymbol{x}$ is formed by $\{r_{nm}\}, \{y_m\}$. Compared to the original form of $\mathcal{P}$, in the standard form we introduce slack variables $\{y_m\}$ to replace inequality constraints with equality constraints. The slack variables essentially capture the difference between the service rates and the total rates of incoming jobs.

Let $\boldsymbol{x} = \{r_{nm}, y_m\}$ be an extreme point of $\mathcal{P}$ in standard form. Under the extreme point, we define s_m to be an idle server if $r_{nm} = 0$ for all $u_n \in \mathcal{U}_m$ and s_m to be a slack server if $y_m > 0$. For a dispatcher u_n and server s_m that are connected, we say that the link between u_n and s_m is *essential*, if $r_{nm} > 0$. In Proposition 2, we show that each extreme point induces a spanning forest of $\mathcal{G}$. Similar results and extreme point structure have also been applied in heavy traffic analysis of stochastic queueing networks [30, 31]. The proof relies on standard techniques for analyzing linear programs related to network flow. We give the proof in **Appendix A** for completeness.

Proposition 2. *Under any extreme point $\{r_{nm}, y_m\}$, the subgraph induced by essential links and idle servers form a spanning forest of the bipartite network $\mathcal{G}$. Each tree in the spanning forest contains at most one slack server. Furthermore, if the extreme point is non-degenerate, then each tree contains exactly one slack server.*

To give an example of Proposition 2, consider the network shown in Figure 3(a) with the essential links of an extreme point marked in red. The values of the $\{r_{nm}\}$ variables are labeled besides the links. In this example, the extreme point is non-degenerate and the only slack server is s_4 (with $y_4 = 2$). The spanning forest induced by the extreme point has two trees: one is a trivial tree only consisting of the idle server s_2, and the other is shown in Figure 3(b).

3.3 Conditions on Network Statistics

We define the following two conditions.

- Condition 1: $\Delta_1 \coloneqq \min_{\boldsymbol{x} \in \mathcal{E}, \boldsymbol{x} \neq \boldsymbol{x}^*} \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x} - \boldsymbol{c}^{\mathrm{T}} \boldsymbol{x}^* > 0$ is a constant independent of T.
- Condition 2: the optimal extreme point is non-degenerate, and every basis is either Δ_2-feasible or Δ_2-infeasible with $\Delta_2 > 0$ being a constant independent of T.

Using the terminology of the online learning literature [11], Δ_1 and Δ_2 can be viewed as the "instance-dependent" parameters, where Δ_1 denotes the gap between the optimal and the second best extreme point, and Δ_2 represents the minimum absolute value of the non-zero basis variables. Note that under Condition 2, as the optimal extreme point is feasible and non-degenerate, all of its basis variables are non-zero, and larger than or equal to Δ_2. Using the structure of extreme points defined in Proposition 2, condition 2 means that for each extreme point (basic feasible solution), the rates of jobs on essential links are at least Δ_2 and the one slack server has at least Δ_2 of extra capacity ($y_m \geq \Delta_2$). For each infeasible basic solution, the rates of jobs on essential links are at least

Δ_2 and the servers with arrival greater than service have $y_m \leq -\Delta_2$ (which is what makes the basic solution infeasible).

For the rest of the paper, we will assume that conditions 1 and 2 hold, and focus on analyzing instance-dependent regret (with Δ_1, Δ_2 being the instance-dependent parameters), where the instance of the optimal control problem is fixed and we study the scaling of the regret with respect to the time horizon T. Note that existing lower bounds on stochastic linear optimization [11, 17] establishes that only in the instance-dependent case can one hope to achieve logarithmic regret, otherwise, it is impossible to have a policy with regret better than $\Omega(\sqrt{T})$.

The standard form of $\mathcal{P}$ was introduced to facilitate the definitions related to the extreme points. For consistency of notations, in what follows we will focus on the original form of $\mathcal{P}$ (instead of the standard form) and use vector $\boldsymbol{r}$ or $\{r_{nm}\}$ to represent a generic feasible solution to $\mathcal{P}$.

4 OVERVIEW AND DISCUSSION OF RESULTS

In this section, we give an overview of our main results and discuss their relation to previous works in the literature.

4.1 Overview of Results

Proposition 1 shows that solving the optimal control problem boils down to learning the solution to $\mathcal{P}$ in the context of stochastic queueing networks and making routing and scheduling decisions based on the learned solution. Therefore, we can break down the optimal control problem into two logical components: the *learning algorithm* and the *control policy*. The learning algorithm (approximately) solves $\mathcal{P}$ using the feedback available in the optimal control problem. The control policy makes routing and scheduling decisions based on the solution provided by the learning algorithm. Note that the two logical components are anything but disjoint. They must be integrated as a joint policy as the learning algorithm updates its solution based on the utility observations, the dynamics of which is determined by the control policy, while the control policy relies on the solution fed by the learning algorithm.

For the control policy, we propose one that relies on the extreme point of $\mathcal{P}$ as input. It relies on the graphical structure, or more specifically, the spanning forest induced by the extreme point rather than the value of the extreme point. This makes it robust to the errors in the solution provided by the learning algorithm. We will show that, given the optimal extreme point, our control policy achieves logarithmic regret. The control policy is a mixture of threshold-based join-the-shortest-queue routing and priority-based routing at the dispatchers, and priority-based scheduling at the servers, with the priority defined by the structure of the forest. More details will be given in Section 5.

The learning algorithm and the joint policy that integrates the learning algorithm and the routing policy will be presented in Section 6. We adapt the algorithm for stochastic linear optimization with bandit feedback proposed in [17] to learn the solution to $\mathcal{P}$. The algorithm from [17] cannot be directly applied here due to the unknown feasibility region of $\mathcal{P}$ (which is parameterized by the unknown network statistics) and the feedback delay. More details will be given regarding these two challenges in Section 6. We will adapt the algorithm from [17] and combine it with our control policy to form a joint learning and control policy and show that it achieves logarithmic regret for the optimal control problem.

4.2 Discussion

In this section, we review related results in the literature and discuss how they fall short of achieving logarithmic regret for the optimal control problem, which will also highlight the novelty of our results on learning in stochastic queueing networks.

4.2.1 Online Learning and Online Decision Making. As we have mentioned, there has been extensive effort in the field of online learning/online decision making dedicated to optimization problems with unknown utility functions with feedback given through (zero-order) oracle on function values [9, 17, 25–27]. Due to the unknown constraints and feedback delay, those algorithms cannot be directly applied to learn the optimal extreme point of $\mathcal{P}$. More importantly, as those works study a pure optimization problem, they are not concerned with the networking aspect of the optimal control problem, i.e., how to translate the learned solution to an effective control policy and how the control policy affects the learning process. Note that many works in the online learning literature consider adversarial objective functions that are time varying [32], which is more general than the fixed utilities we consider in the optimal control problem. However, that generality does not help as their settings still do not involve stochastic queueing dynamics.

4.2.2 Network Utility Maximization. The optimal control problem can be considered as a problem of network utility maximization with (unknown) linear utility function. Although results for general network utility maximization problems [8, 19, 20, 23, 24, 28] can be used to derive viable policies for the optimal control problem, we will justify in the following that those policies will only achieve $\Omega(\sqrt{T})$-regret, which is strictly worse than the logarithmic regret achieved by our policy.

Consider the simple network with one dispatcher and two servers in Figure 2. Since $v_{11} = 6 > v_{12} = 5$, the optimal solution is $r^*_{11} = 5, r^*_{12} = 2$, and the network statistics satisfy conditions 1 and 2 with $\Delta_1 = \Delta_2 = 3$. We first consider a simplistic case where the optimal solution is given and we only need a good control policy to achieve low regret. A simple idea is to use a static randomized routing policy parameterized by the optimal solution combined with an arbitrary scheduling policy. For example, a valid static policy based on $\{r^*\}$ is one that at each time sends the incoming jobs to s_1 with probability $\frac{5}{7}$ and to s_2 with probability $\frac{2}{7}$ while the servers serve the jobs in an arbitrary order. This policy seems natural, but it has a utility gap of $\Omega(\sqrt{T})$, which is defined as the difference between the expected utility achieved by the policy and T times the value of the solution $\boldsymbol{r}^*$, and this will lead to $\Omega(\sqrt{T})$-regret. The reason is that the queue of s_1 is critically loaded, which will result in $\mathbb{E}[Q_1(T)] = \Omega(\sqrt{T})$ and cause a $\Omega(\sqrt{T})$-loss of utility[2]. This is by no means specific to the static randomized policy considered in the example, as we will formally show in Appendix B that any static policy that makes routing decisions independently over time has a regret of $\Omega(\sqrt{T})$, which is inferior to the logarithmic regret that our policy achieves. As existing works on network utility maximization use variants of Max-Weight policies which are derived from minimizing certain quadratic Lyapunov function that seek to converge to the optimal static policy, they also cannot achieve a regret better than $\Omega(\sqrt{T})$ [19]. It follows that the network control component alone is already non-trivial. Furthermore, when we bring the problem of learning the optimal solution back into the picture, we see that to achieve logarithmic regret, the control policy cannot rely on the values of the solution. Since both the objective function and the feasibility region of the static linear program $\mathcal{P}$ are unknown, existing lower bounds [11, 17] establish that it is impossible to obtain the solution to $\mathcal{P}$ within an error smaller than $\Theta(\frac{1}{\sqrt{T}})$ after T time slots. For example, in the aforementioned case (Figure 2), no learning algorithm can obtain the exact values of $r^*_{11} = 5, r^*_{12} = 2$ but at best $r^*_{11} \simeq 5 \pm \Theta(\frac{1}{\sqrt{T}}), r^*_{12} \simeq 2 \pm \Theta(\frac{1}{\sqrt{T}})$. Therefore, relying on the values of the solution will inevitably lead to $\Omega(\sqrt{T})$-regret. To achieve logarithmic regret, we have to rely on some structure of the solutions that is robust against the error.

4.2.3 Reinforcement Learning. The optimal control problem can be formulated as a Markov decision process with unknown parameters. Therefore, reinforcement learning techniques can also be applied.

[2]This follows because the queue length of a critically loaded queue grows at the order of $\Theta(\sqrt{T})$ with time.

However, none of the existing results on reinforcement learning can be shown to achieve a regret better than $O(\sqrt{T})$ [12–16]. Due to the generality of the reinforcement learning framework, results therein cannot exploit the structure of the optimal control problem.

5 THE DUAL-LEVEL JSQ-K POLICY

In this section, we introduce the control policy for the optimal control problem, which we call the Dual-Level JSQ-K policy. At the dispatchers, it makes routing decisions based on the priority levels defined on the forest induced by the extreme point in a Join-the-Shortest-Queue fashion with queue length threshold K. At the servers, it makes priority-based scheduling decisions also based on the priority levels defined on the forest. In what follows, we first introduce the priority levels of extreme points, and then present the details and performance analysis of the Dual-Level JSQ-K policy.

5.1 Priority Levels Induced by Extreme Point

Given an extreme point of the linear program $\mathcal{P}$, we associate a priority level with each server and dispatcher as follows. For a tree in the spanning forest induced by the extreme point, if there is a slack server in the tree, then we designate the slack server as the root, otherwise, we designate an arbitrary server in the tree as the root. Based on the designated root, we essentially give each tree an orientation. For each node in the tree, we define its priority level as its distance to the root in the tree. For example, the root server has priority level 0, and the job dispatchers that are immediately connected to the root server in the tree have priority level 1. From these definition, we have the following observations:

- For a dispatcher of level h, it is connected to exactly one server of level $h-1$. If the dispatcher is not a leaf node, it is also connected to at least one server of level $h+1$. We will refer to the level $h-1$ server as the secondary server of the dispatcher, and the level $h+1$ server(s) as the primary server(s) of the dispatcher.
- For a server of level $h \neq 0$, it is connected to exactly one dispatcher of level $h-1$. If the server is not a leaf node, it is also connected to at least one dispatcher of level $h+1$. We will refer to the level $h-1$ dispatcher as the secondary dispatcher of the server, and the level $h+1$ dispatcher(s) as the primary dispatcher(s) of the server.
- If a server s_m is a primary server of a dispatcher u_n, then the dispatcher u_n is the (only) secondary dispatcher of the server s_m. Similarly, if s_m is the secondary server of a dispatcher u_n, then u_n is a primary dispatcher of s_m.

5.2 The Control Policy

We now present the details of the dual-level JSQ-K policy, which we will often abbreviate as the JSQ-K policy. The JSQ-K policy is parameterized by a threshold parameter K whose value will be set later. For a given extreme point $\boldsymbol{r}$, the JSQ-K policy is structured based on the spanning forest induced by $\boldsymbol{r}$.

Scheduling: The queue of each server s_m is partitioned into two virtual queues Q_m^h and Q_m^l. Under an extreme point, the virtual queue Q_m^h is the high-priority queue that holds the jobs from the primary dispatchers of the server while the virtual queue Q_m^l is the low-priority queue that holds the jobs from the secondary dispatcher of the server. As an example, the virtual queueing architecture corresponding to the extreme point in the example of Figure 3(a) is shown in Figure 3(b). For each server, the scheduling policy is to gives priority service to the jobs in its high-priority queue and only serves the jobs in its low-priority queue if its high-priority queue is empty.

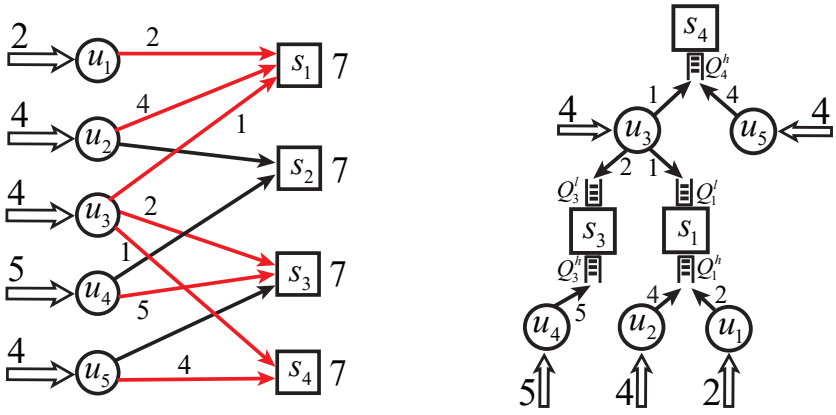

Fig. 3. Illustration of the priority levels and the virtual queue architecture induced by an extreme point.

Routing: For each dispatcher u_n that is connected to its primary servers $m_1, \ldots, m_L$ and secondary server m_0, it first checks if any of its primary servers have low-priority queue with length no larger than the threshold K. If so, then it sends the jobs to the primary server with the smallest low-priority queue length. Otherwise, it then checks if the high-priority queue of its secondary server is no larger than K. If so, it sends the incoming jobs to its secondary server. If all the low-priority queues of $m_1, \ldots, m_L$ and the priority queue of m_0 are greater than K, then the dispatcher discards the incoming jobs[3]. The pseudo-code of the JSQ-K policy is shown in **Algorithm 1**.

To give a concrete example, consider the extreme point in Figure 3(b). On the server side, the priority queue of server s_1 receives jobs from dispatchers u_1 and u_2. The low-priority queue of s_1 receives jobs from u_3. The high-priority queue of s_3 receives jobs from u_4 and the low-priority queue of s_3 receives jobs from u_3. The high-priority queue of s_4 receives jobs from u_3 and u_5. On the dispatcher side, both u_1 and u_2 only sends jobs to s_1 when $Q_1^h \leq K$, and discard the incoming jobs otherwise. Dispatcher u_4 sends jobs to s_3 when $Q_3^h \leq K$ and discard the incoming jobs otherwise. Dispatcher u_3 sends jobs to the shorter of Q_1^l, Q_3^l when at least one of them is no larger than K, otherwise u_3 sends jobs to s_4 when $Q_4^h \leq K$. When Q_1^l, Q_3^l, Q_4^h are all greater than K, u_3 discards the incoming jobs. Dispatcher u_5 sends jobs to s_4 when $Q_4^h \leq K$, and discards the incoming jobs otherwise.

5.3 Analysis

We first show the claim that given the optimal extreme point, the JSQ-K policy achieves logarithmic regret. The claim will follow from Theorem 1, which establishes a more general statement: given any non-degenerate extreme point $\boldsymbol{r}$, the difference between the total utility achieved by the JSQ-K policy given $\boldsymbol{r}$ as input and $T \cdot U(\boldsymbol{r})$ is in $O(\log T)$, where $U(\boldsymbol{r}) = \sum_{n,m} v_{nm} r_{nm}$ is the value of $\boldsymbol{r}$ with respect to the objective function of $\mathcal{P}$. Combining Proposition 1 and Condition 2, we have that the optimal extreme point is non-degenerate. Hence, we have that Theorem 1 implies the claim that the JSQ-K policy achieves logarithmic regret if given the optimal extreme point.

Theorem 1. *For any non-degenerate extreme point $\boldsymbol{r}$, the total expected utility achieved by the JSQ-K policy based on the spanning forest induced by $\boldsymbol{r}$ with $K = O(\log T)$ is at least $T \cdot U(\boldsymbol{r}) - O(\log T)$.*

[3]The policy would still work if the dispatcher makes an arbitrary routing decision in this case. We present the job-discarding version for convenience of analysis.

Algorithm 1 The Dual-Level JSQ-K Policy

Input: Parameter K

Initialize: $Q^h_m(0) = 0, Q^l_m(0) = 0$ for each m.
for $t = 0, 1, \ldots,$ **do**
  for each server s_m **do**
    Gives priority service to jobs in $Q^h_m(t)$.
  for each dispatcher u_n with primary servers $m_1, \ldots m_l$ and secondary server m_0 **do**
    if there exists $i \in \{1, \ldots, L\}$ with $Q^l_{m_i}(t) \le K$ **then**
      $i^* := \arg\min_{i \in \{1,\ldots,L\}} Q^l_{m_i}(t)$.
      Send the jobs from u_n to $Q^l_{m_{i^*}}$. $a_{nm_{i^*}}(t) = a_n(t)$.
    else if $Q^h_{m_0}(t) \le K$ **then**
      Send the jobs from u_n to $Q^h_{m_0}$. $a_{nm_0}(t) = a_n(t)$.
    $a_{nm}(t) := 0$ for all other servers m.
  Update $Q^h_m(t), Q^l_m(t)$ for each m.

Due to the space limitations, we give the overall structure and the intuition of the proof here. The proof details are deferred to **Appendix C**.

The proof of Theorem 1 consists of proving two claims for each node in the spanning forest. Under a policy π, we define the random variable $\tilde{c}^\pi_{nm}(t)$ as the number of jobs from dispatcher u_n completed at server s_m at time t. In the proof, we will omit the superscript π as it will always refer to the JSQ-K policy. Consider a non-degenerate extreme point $\boldsymbol{r}$. The extreme point induces a spanning forest of $\mathcal{G}$. We consider an arbitrary tree in the spanning forest with $H+1$ priority levels $0, \ldots, H$. We will establish two claims for each dispatcher and two claims or each server. More specifically, define $\epsilon_h = \frac{1}{[4(M+N)]^{H-h}}$ for $h = 1, \ldots, H$ and $C_1 = \frac{4(M+N)}{\Delta_2}$. Set $K = 8(4(M+N))^H \log T$. For each dispatcher u_n **at level** $H-h$ with primary servers $m_1, \ldots, m_L$ and secondary server m_0, we establish the following two claims:

Claim (1.1): Starting from $t = C_1 h \ln T$, the probability that there exists a primary server of u_n with queue length (high-priority queue plus low-priority queue) smaller than $(1-\epsilon_h)K$ is small, i.e., $\sum_{t=C_1 h \ln T}^{T} \mathbb{P}[\exists i = 1, \ldots, L, Q^h_{m_i}(t) + Q^l_{m_i}(t) \le (1-\epsilon_h)K] = O(\frac{1}{T})$.

Claim (1.2): The expected total completed service from u_n at each of the primary server m is close to $T \cdot r_{nm}$, i.e., for each $i = 1, \ldots, L$, $Tr_{nm_i} - O(\log T) \le \sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{nm_i}(t)] \le Tr_{nm_i} + O(\log T)$.

For each server s_m (of priority level $H-h$) with primary dispatchers $n_1, \ldots, n_L$ and secondary dispatcher n_0, we establish the following two claims

Claim (2.1): Starting from $t = C_1 h \ln T$, the probability that the high-priority queue of s_m grows over $\epsilon_h K$ is small, i.e., $\sum_{t=C_1 h \ln T}^{T} \mathbb{P}[Q^h_m(t) \ge \epsilon_h K] = O(\frac{1}{T})$.

Claim (2.2): The expected total completed service at s_m from each of its primary dispatcher u_n is close to $T \cdot r_{nm}$, i.e., for each $i = 1, \ldots, L$, $Tr_{n_i m} - O(\log T) \le \sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{n_i m}(t)] \le Tr_{n_i m} + O(\log T)$.

Note that the constant ϵ_h decreases as the level of the node increases (going from the root to leaf nodes). Claim (1.1) shows that for each server m_i that is not the root (slack) server, its queue length rarely goes below $(1-\epsilon_h)K$ after $O(\log T)$ time slots, which implies that it is almost never idle. This also implies that non-slack servers are fully utilized. Claim (2.1) shows that for each server, its high-priority queue is rarely greater than $\epsilon_h K$ with $\epsilon_h < 1$. This implies that dispatchers almost never drop the incoming jobs since the high-priority queue of their secondary server is almost always smaller than K. Claims (1.1) and (2.1) are intermediate steps that are instrumental in proving Claims (1.2) and (2.2). After having proved Claim (1.2) for each dispatcher in the tree and Claim

(2.2) for each server in the tree, it will follow that the difference between the total expected utility obtained under the policy and $T \cdot U(\boldsymbol{r})$ is $O(\log T)$, which will imply Theorem 1.

The proof of the claims with respect to each node in the tree is done via an induction framework.

Structure of the Induction: For each tree in the spanning forest, the base step of the induction deals with the nodes at level $H-1$ (parents of the leaf nodes). The base step starts from nodes at level $H-1$ instead of nodes at level H because nodes at level H (the leaf nodes) do not have any children and thus their corresponding claims trivially hold. Depending on whether the nodes are servers or dispatchers, the base step can be divided into two cases. In the first case, the nodes at level $H-1$ are dispatchers (and the leaf nodes are servers) and we need to prove Claim (1.1) and Claim (1.2) for each dispatcher at level $H-1$. In the second case, the nodes at level $H-1$ are servers (and the leaf nodes are dispatchers) and we need to prove Claim (2.1) and Claim (2.2) for each server at level $H-1$. Proceeding from the base step, the induction step works by proving for each node in the tree Claims (1.1), (1.2) if the node is a dispatcher, and Claims (2.1), (2.2) if the node is a server, assuming the induction hypothesis that Claims (1.1), (1.2) hold for all the dispatchers, and Claims (2.1), (2.2) hold for all the servers in the subtree rooted at the node. When completing the induction, we will have proved the corresponding claims of each node in the forest.

Intuition Behind the Claims: Now we give the intuition behind why the claims hold. The details of establishing the claims are deferred to **Appendix C**. We first give the main intuition behind Claims (1.1) and (1.2). Consider a dispatcher u_n whose primary servers $m_1, \ldots, m_L$ are the leaf nodes of the tree and secondary server is m_0. For Claim (1.2), note that in this case $r_{nm_i} = \mu_{m_i}$. Therefore, the upper bound of $\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{nm_i}(t)]$ is straightforward. While for the lower bound, since each primary server only receives jobs from u_n, we need to show that the cumulative idleness in each of the servers $m_1, \ldots, m_L$ is in $O(\log T)$, which will essentially follow from Claim (1.1). For Claim (1.1), note that if we consider servers $m_1, \ldots, m_L$ as a set, as long as one of the servers have queue length smaller than K, the incoming jobs from u_n (which is of rate λ_n) are sent to the set while the total service rate of the set is $\sum_{i=1}^{L} \mu_i$. By the constraint satisfied by the extreme point, $\lambda_n - \sum_{i=1}^{L} \mu_i = r_{nm_0} > 0$. Therefore, the total queue lengths of the set tend to have positive drift when the queues are not too large, from which Claim (1.1) can be derived. When proving Claims (1.1) and (1.2) for a dispatcher higher up in the tree (i.e. the induction steps), the key ideas are the same but the drift arguments are more challenging to construct since the queueing dynamics will be influenced by the servers and dispatchers of higher priorities (that are descendents of the dispatcher in the tree).

For the main intuition behind Claims (2.1) and (2.2), consider a server s_m whose primary dispatchers $n_1, \ldots, n_L$ are the leaf nodes and secondary dispatcher is n_0. In this case $r_{n_i m} = \lambda_{n_i}$. Hence for Claim (2.2) the upper bound on $\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{n_i m}(t)]$ is straightforward. For the lower bound, since each dispatcher $n_1, \ldots, n_L$ only sends jobs to s_m, we need to show that the queue length of s_m and the total number of jobs discarded are in $O(\log T)$. The queue length of s_m is in $O(\log T)$ by design as $K = O(\log T)$. The total number of jobs discarded is the same order as the total probability of Q_m^h being greater than K over the whole time horizon which will essentially follow from Claim (2.1). For Claim (2.1), note that the total arrival rate to Q_m^h is at most $\sum_{i=1}^{L} \lambda_{n_i}$, while the offered service rate to Q_m^h (as it receives priority service) is μ_m as long as $Q_m^h > 0$. As by the constraint satisfied by the extreme point, $\mu_m - \sum_{i=1}^{L} \lambda_{n_i} = r_{n_0 m} > 0$, Q_m^h is a queue with negative drift, from which Claim (2.1) can be derived. Again, when proving Claims (2.1) and (2.2) for a server higher up in the tree (i.e. the induction steps), we use the same ideas but need to be more careful when constructing the drift arguments.

6 LEARNING ALGORITHM AND THE JOINT POLICY

In this section, we will first present our learning algorithm, which is an adaptation of the Confidence-Ball algorithm proposed in [17]. Next, we integrate the learning algorithm with the JSQ-K policy to form a joint policy and prove that it achieves logarithmic regret for the optimal control problem.

6.1 Learning Algorithm: The Confidence-Ball Algorithm

6.1.1 Stochastic Linear Optimization with Bandit Feedback. We start by briefly reviewing the result of [17]. The work [17] studied the problem of stochastic linear optimization with bandit feedback. Consider the linear optimization problem $\max_{\boldsymbol{r}\in D} \boldsymbol{v}\cdot\boldsymbol{r}$, where $D\in\mathbb{R}^k$ is the feasibility region that is known in advance but $\boldsymbol{v}$ is unknown. At every time t, we choose a decision vector $\boldsymbol{r}_t\in D$ and receives an observation $l_t=\boldsymbol{v}\cdot\boldsymbol{r}+\epsilon_t$, where ϵ_t is a zero-mean noise with bounded variance. Let the optimal solution be $\boldsymbol{r}^*\coloneqq\arg\max_{\boldsymbol{r}\in D}\boldsymbol{v}\cdot\boldsymbol{r}$. The goal is to design an algorithm that outputs a sequence of decisions $\boldsymbol{r}_1,\ldots,\boldsymbol{r}_T$ such that the regret $R_T=\sum_{t=1}^T(\boldsymbol{v}\cdot\boldsymbol{r}_t^*-\boldsymbol{v}\cdot\boldsymbol{r}_t)$ is low.

The algorithm proposed by [17], the Confidence-Ball algorithm, achieves logarithmic regret for stochastic linear optimization with bandit feedback. Before reviewing the details of the algorithm, the following definitions are needed. For a vector $\boldsymbol{\nu}\in\mathbb{R}^n$ and a positive definite matrix $A\in\mathbb{R}^{n\times n}$, we denote $||\boldsymbol{\nu}||_{1,A}\coloneqq||A^{1/2}\boldsymbol{\nu}||_1=\sum_{i=1}^n|A^{1/2}\boldsymbol{\nu}|_i$ as the 1-norm based on A. The details of the algorithm is shown in **Algorithm 2**. The algorithm essentially works through estimating the vector $\boldsymbol{v}$ from the observations using a linear regression-like procedure. Note that if we are given $(\boldsymbol{r}_1,l_1),\ldots,(\boldsymbol{r}_t,l_t)$, the problem of estimating $\boldsymbol{v}$ resembles the linear regression problem, where the estimate is given by $\hat{\boldsymbol{v}}_t\coloneqq(\sum_{\tau=1}^t\boldsymbol{r}_\tau\boldsymbol{r}_\tau')^{-1}\sum_{\tau=1}^t l_\tau\boldsymbol{r}_\tau$. The Confidence-Ball algorithm essentially uses the same procedure, where the matrix A_t keeps track of $\sum_{\tau=1}^t\boldsymbol{r}_\tau\boldsymbol{r}_\tau'$ but initialized with the Bary-centric spanner to make sure that A_t is invertible for $t=1,\ldots,T$. Instead of using the point estimate $\hat{v}_t$, the algorithm uses the best $\boldsymbol{v}$ in an ellipsoid (confidence-ball) around $\hat{v}_t$ (Line 4) to solve for $\boldsymbol{r}_t$ (Line 5). Note that as pointed out in [17], when D is a polyhedron, every $\boldsymbol{r}_t$ can be solved to be an extreme point of D.

Algorithm 2 The Confidence-Ball Algorithm

1: **Initialization:** Barycentric spanner $\boldsymbol{b}_1,\ldots,\boldsymbol{b}_k$ for D; $A_1=\sum_{i=1}^k\boldsymbol{b}_i\cdot\boldsymbol{b}_i'$; $\hat{\boldsymbol{v}}_1=0$.
2: **for** $t=1,\ldots,T$ **do**
3: $\quad\beta_T=512k\ln^2 T$.
4: $\quad B_T^1=\{\boldsymbol{\nu}:||\boldsymbol{\nu}-\hat{\boldsymbol{v}}_t||_{1,A_t}\le\sqrt{k\beta_T}\}$.
5: $\quad\boldsymbol{r}_t=\arg\max_{\boldsymbol{x}\in D}\max_{\boldsymbol{v}\in B_t^1}(\boldsymbol{v}'\cdot\boldsymbol{r})$.
6: $\quad$Receives unbiased observation of the objective function $l_t\coloneqq\boldsymbol{v}\cdot\boldsymbol{r}_t+\epsilon_t$.
7: $\quad A_{t+1}=A_t+\boldsymbol{r}_t\cdot\boldsymbol{r}_t'$.
8: $\quad\hat{\boldsymbol{v}}_{t+1}=A_{t+1}^{-1}\sum_{\tau=1}^t l_\tau r_\tau$.

6.1.2 Challenges of Applying the Confidence-Ball Algorithm. Two main challenges prevent us from directly applying the Confidence-Ball Algorithm to find the optimal extreme point of the static linear program $\mathcal{P}$ in the optimal control problem. The first one is the unknown feasibility region. Recall that the feasibility region D of $\mathcal{P}$ is written as $D=\{\{r_{nm}\}\mid\sum_{m\in\mathcal{S}_n}r_{nm}=\lambda_n,\sum_{n:s_m\in\mathcal{S}_n}r_{nm}\le\mu_m,r_{nm}\ge0.\}$. The set D is unknown apriori since it is parameterized by unknown network statistics (arrival rates and service rates).

The second challenge arises from the delay in obtaining unbiased estimate of the objective function. The Confidence-Ball algorithm requires an unbiased estimate of the objective function $\boldsymbol{v}\cdot\boldsymbol{r}_t$ for a decision vector $\boldsymbol{r}_t$. Such unbiased estimate is not directly available in the optimal control problem, but can be synthesized from utility observations of completed jobs. Consider a decision

vector $\boldsymbol{r} = \{r_{nm}\}$ for $\mathcal{P}$. If for each u_n, s_m with $r_{nm} > 0$, we have a utility observation $\hat{v}_{nm}$ from a job from u_n completed at s_m (which from now on will be referred to as a (u_n, s_m)-job), then $\sum_{n,m:r_{nm}>0} \hat{v}_{nm} \cdot r_{nm}$ is an unbiased estimate of the objective function of $\mathcal{P}$ at $\{r_{nm}\}$. However, such synthesized estimates are not immediately available since we only observe utilities upon the completion of the jobs which experience queueing delay.

Therefore, the Confidence-Ball algorithm cannot be directly applied in the optimal control problem. In what follows, we will propose an adapted version of the Confidence-Ball algorithm that address the aforementioned two challenges and integrate the algorithm with the JSQ-K policy.

6.2 The Confidence-Ball JSQ-K Policy

In this section, we propose the an adapted version of the Confidence-Ball algorithm and integrate it with the Dual-Level JSQ-K policy to form the Confidence-Ball JSQ-K policy, which will be shown to achieve logarithmic regret for the optimal control problem.

6.2.1 The Adapted Confidence-Ball Algorithm. We propose the Adapted Confidence-Ball algorithm to deal with the unknown feasibility region of the static linear program $\mathcal{P}$. Note that for each time t, we can observe the arrivals $a_n(t)$'s and offered services $c_m(t)$'s. Thus, we can form an estimation of the feasibility region D at each time t based on the empirical means of the arrivals and services.

Definition 1. *Let* $\hat{\lambda}_t := \frac{1}{t}\sum_{\tau=1}^{t} a_n(\tau), \hat{\mu}_m^t := \frac{1}{t}\sum_{\tau=1}^{t} c_m(\tau)$. *The estimation of the feasibility region at time* t *is defined as* $\hat{D}_t := \{\{r_{nm}, y_m\} \mid \sum_{m\in\mathcal{S}_n} r_{nm} = \hat{\lambda}_n^t, \sum_{n:s_m\in\mathcal{S}_n} r_{nm} \le \hat{\mu}_m^t, r_{nm} \ge 0.\}$.

The Adapted Confidence-Ball algorithm is the same as **Algorithm 2** but with Line 5 replaced with minimization over $\hat{D}_t$, i.e., $\boldsymbol{r}_t = \arg\max_{\boldsymbol{r}\in\hat{D}_t} \max_{\boldsymbol{v}\in B_t^1} (\boldsymbol{v}' \cdot \boldsymbol{r})$, and Line 6 replaced with synthesizing the unbiased estimate using corresponding utility observations as introduced in Section 6.1.2.

We define $\boldsymbol{r}_t^*$ as the optimal extreme point of $\hat{D}_t$, i.e., $\boldsymbol{r}_t^* := \arg\max_{\boldsymbol{r}\in\hat{D}_t} \boldsymbol{v}\cdot\boldsymbol{r}$. Using the results of [17], we have the following proposition regarding the performance guarantee of the Adapted Confidence-Ball algorithm.

Proposition 3. *With probability at least* $1 - \frac{1}{T}$, *during* T *iterations of the Adapted Confidence-Ball algorithm, the number of* t *in* $\{1, \dots, T\}$ *such that* $\boldsymbol{r}_t \neq \boldsymbol{r}_t^*$ *is in* $O(\log^3 T)$.

6.2.2 Integration of Adapted Confidence-Ball and Dual-Level JSQ-K. Now we are ready to introduce the Confidence-Ball JSQ-K (CB-JSQ-K) policy, which the joint policy that integrates the Adapted Confidence-Ball algorithm and the Dual-Level JSQ-K policy for the optimal control problem.

The basic idea of the CB-JSQ-K policy is to make routing decisions using the JSQ-K policy based on the extreme-point solution provided by the Adapted Confidence-Ball algorithm. Due to the aforementioned feedback delay, the Adapted Confidence-Ball algorithm cannot update the solution every time slot. Instead, we employ an episodic version of the Adapted Confidence-Ball algorithm where the solution is updated once every episode (consisting of multiple time slots). The routing decisions are made using JSQ-K based on the same solution during each episode. The episode length will be set long enough to ensure that the Adapted Confidence-Ball algorithm can obtain the utility observations necessary to synthesis unbiased estimates of the objective function.

More specifically, the CB-JSQ-K policy embeds the JSQ-K policy in an episodic version of the Adapted Confidence-Ball algorithm. The episode length is set to $W = \log^2 T \log\log T$. We index the episodes by $e = 1, \dots$ and let t_e be the first time slot of episode e. The policy maintains the matrix A_e, vector estimate $\hat{\boldsymbol{v}}_e$ of $\boldsymbol{v}$, and ellipsoid B_e around $\hat{\boldsymbol{v}}_e$ for every episode e. At the beginning of each episode, it solves for an extreme point $\boldsymbol{r}^e$ through optimization over B_e and $\hat{D}_e$, where $\hat{D}_e$ is the estimation of feasibility region (Definition 1) at the beginning of episode e. Let $\mathcal{T}_e$ be the

spanning forest induced by $\boldsymbol{r}^e$. If $\mathcal{T}_e \neq \mathcal{T}_{e-1}$, the policy discards all the jobs in the queues.[4] Then, it makes routing and scheduling decisions based on the JSQ-K policy using $\mathcal{T}_e$, while collecting utility observations. At the end of the episode, the policy updates $\hat{D}_e$ based on the observations of arrivals and offered services, and updates $A_e, \hat{\boldsymbol{v}}_e$ if we have obtained at least one utility observation of a (u_n, s_m)-job for each $r_{nm} > 0$. The pseudo-code of the CB-JSQ-K policy is shown in **Algorithm 3**.

Algorithm 3 The Confidence-Ball JSQ-K Policy

Initialization: Barycentric spanner $b_1, \ldots, b_n$ for $\hat{D}_1$; $A_1 = \sum_{i=1}^{k} b_1 b_1^{\mathrm{T}}$; $\hat{\boldsymbol{v}}_1 = 0$; $W = \log^2 T \log\log T$.
$\beta_T = 512k \ln^2 T$.
for episode $e = 1, \ldots$ **do**
  $B_e^1 = \{\boldsymbol{v} : ||\boldsymbol{v} - \hat{\boldsymbol{v}}_e||_{1,A_e} \leq \sqrt{k\beta_T}\}$.
  $\boldsymbol{r}^e := \arg\max_{\boldsymbol{r} \in \hat{D}_e} \max_{\boldsymbol{v} \in B_e^1} (\boldsymbol{v}^{\mathrm{T}} \boldsymbol{r})$.
  $\mathcal{T}_e :=$ the spanning forest induced by $\boldsymbol{x}^e$.
  Discard all the jobs in the queues if $\mathcal{T}_e \neq \mathcal{T}_{e-1}$.
  for $t = t_e, t_e + 1, \ldots, t_e + W - 1$ **do**
    Make routing and scheduling decisions based on the JSQ-K policy on the forest $\mathcal{T}_e$.
    When the first (u_n, s_m) job of the episode is completed with utility observation $\hat{v}_{nm}$, set $\hat{v}_{nm}^e := \hat{v}_{nm}$.
  Collect the observations of arrivals and offered services during the episode and update $\hat{D}_e$ to $\hat{D}_{e+1}$.
  if at least one observation is obtained for each (u_n, s_m) with $r_{nm}^e > 0$ **then**
    $A_{e+1} = A_e + \boldsymbol{r}^e \cdot (\boldsymbol{r}^e)'$.
    $l_e := \sum_{n,m: r_{nm}^e > 0} \hat{v}_{nm}^e r_{nm}^e$.
    $\hat{\boldsymbol{v}}_{e+1} = A_{e+1}^{-1} \sum_{\tau=1}^{e} l_\tau \boldsymbol{r}_\tau$.
  else
    $A_{e+1} := A_e$, $l_e := 0$, $\hat{\boldsymbol{v}}_{e+1} := \hat{\boldsymbol{v}}_e$.

6.2.3 Analysis of CB-JSQ-K. We show in Theorem 2 that the Confidece-Ball JSQ-K policy with $K = O(\log T)$ achieves logarithmic regret.

Theorem 2. *The CB-JSQ-K policy achieves $O(\log^5 T \log\log T)$ regret with an appropriately chosen $K = O(\log T)$.*

Proof Sketch: we provide the main idea of the proof here and defer the details to **Appendix D**. The first step of the proof is to show that we will be able to obtain all the necessary utility observations for each episode (with high probability) after $t \geq \log^3 T$. This holds since after $t \geq \log^2 T$, $\hat{\lambda}_n^t, \hat{\mu}_m^t$ will be sufficiently close to λ_n, μ_m so that the estimated feasibility region $\hat{D}_e$ is sufficiently close to the true feasibility region D. It follows that the forest induced by any extreme point of $\hat{D}_e$ will be feasible with respect to D. Therefore, we can construct similar drift argument as the proof of Theorem 1 to show that within each episode e, at least one (u_n, s_m)-job is completed for every $r_{nm}^e > 0$ with high probability, which implies that all the necessary utility observations are obtained. The second step is to show that after $t \geq \log^2 T$, the optimal extreme point of $\hat{D}_e$ and the optimal extreme point of D induces the same spanning forest. This again follows from that $\hat{\lambda}_n^t, \hat{\mu}_m^t$ being sufficiently close to λ_n, μ_m. Combining this with Proposition 3, it implies that there are at most

[4]This step is made for simplicity of analysis, since a different spanning forest may lead to a different virtual queueing architecture.

$O(\log^3 T)$ episodes where the spanning forest used by the JSQ-K policy is sub-optimal. Based on the previous two steps, we proceed to analyze the regret of the CB-JSQ-K policy. We divide the time horizon into periods, where each period is formed by consecutive episodes with the same forest. The regret of the CB-JSQ-K policy is the sum of the regret over each period. We will call a period/episode correct if the spanning forest of the period/episode coincides with the optimal, and a period/episode incorrect otherwise. Since there are at most $O(\log^3 T)$ episodes where $\mathcal{T}_e$ is not equal to the optimal forest, there are at most $O(\log^3 T)$ periods. The total length of incorrect period is upper bounded by the total number of incorrect episodes times the episode length, which is $O(\log^5 T \log\log T)$ time slots. Therefore, the regret incurred in incorrect period is in $O(\log^5 T \log\log T)$. Whenever the policy switches between periods, it discards all the jobs in the queue, which will in total incur $O(\log^4 T)$-regret as the total queue length is in $O(\log T)$. Finally, by Theorem 1, as the optimal extreme point is non-degenerate, the regret incurred in each correct period is $O(\log T)$. Therefore, in summary, the regret of the CB-JSQ-K policy is in $O(\log^5 T \log\log T)$.

Discussion: The main reason that the Confidence-Ball JSQ-K policy can achieve (poly)-logarithmic regret instead of $O(\sqrt{T})$ regret is that the Confidence-Ball JSQ-K policy tries to learn the optimal spanning forest, i.e., the structure of the optimal solution to the linear program $\mathcal{P}$ instead of the value of the optimal solution. Since the total number of extreme points is finite, when the assumptions are satisfied, there is enough "separation" between different extreme points while such separation does not exist for the value of the solution as it lies in a continuous set. This is the main reason why learning the optimal structure is more robust than learning the optimal value.

7 SIMULATIONS

In this section, we evaluate the empirical performance of our policies via simulations on the network shown in Figure 3(a). The arrival rates and service rates are shown in the figure. The underlying utilities are chosen such that the extreme point shown in Figure 3(b) is the optimal extreme point.[5] For each dispatcher u_n, $a_n(t)$ is chosen as a uniform integer between $\lambda_n - 2$ and $\lambda_n + 2$ and for each server s_m, $c_n(t)$ is chosen as a uniform integer between $\mu_m - 2$ and $\mu_m + 2$.

We first study the growth of regret with the time horizon. We vary the time horizon T in $\{10000, 20000, \ldots, 200000\}$ and compare the performance of three policies:

- **Static:** the static optimal control policy based on the optimal solution to $\mathcal{P}$.
- **JSQK:** the Dual-Level JSQ-K policy with $K = 20\log T$ on the optimal extreme point.
- **CB-JSQK:** the Confidence-Ball JSQ-K policy with $K = 20\log T$ and episode length $E = \log^3 T$.

The regret of each policy is approximated by the difference between $T \cdot OPT(\mathcal{P})$ and the total utility obtained over the time horizon. We plot the regret and the total queue lengths at the end of the time horizon in **Figure 4**. The results are the average over 10 runs.

From Figure 4(a), we can see that JSQK and CB-JSQK have significantly lower regrets than the static optimal policy, which validates our theoretical analysis, as the former two achieve logarithmic regret while the latter has a regret of $\Omega(\sqrt{T})$. The regret of CB-JSQK is higher than JSQK since it needs to learn the optimal extreme point while JSQK is fed with the optimal extreme point. Similar phenomenon can be observed for queue lengths in Figure 4(b). However, the queue length under CB-JSQK is slightly lower than JSQK, which can be attributed to that the CB-JSQK policy clears the queues when the current extreme point changes between episodes.

Next, we study the sensitivity of the performance of JSQK and CB-JSQK to the parameter K. We fix the time horizon to be $T = 100000$ and vary K in $\{20, 40, \ldots, 200\}$. We plot the regret and the total queue lengths at the end of the time horizon in **Figure 5**. From **Figure 5**, we can see that for both JSQK and CB-JSQK, the total queue lengths increase with K, which is not surprising given the

[5]More specifically, $v_{11} = 5$, $v_{21} = 5$, $v_{22} = 2$, $v_{31} = 3$, $v_{33} = 3$, $v_{34} = 2$, $v_{42} = 2$, $v_{43} = 4$, $v_{53} = 2$, $v_{54} = 4$.

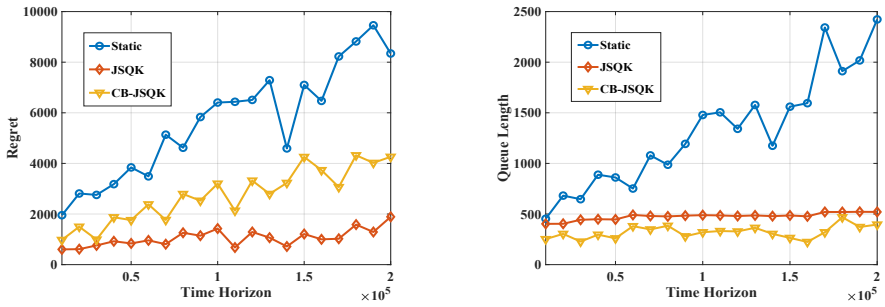

Fig. 4. Progression of regret and queue length with the time horizon under different policies.

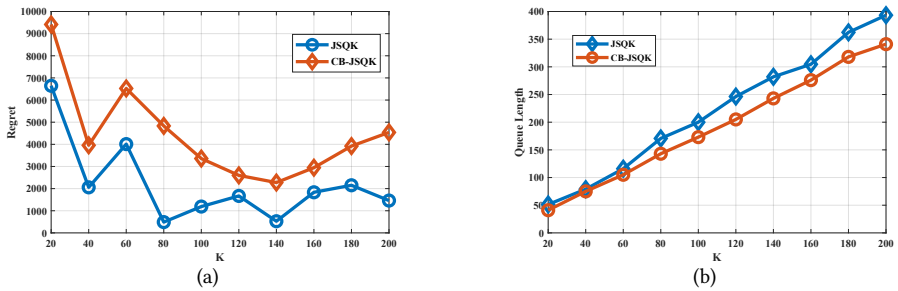

Fig. 5. Regret and queue length of under JSQK and CB-JSQK policies with different values of K.

role of K as the queue length threshold in the policies. Furthermore, the regrets of both policy are not sensitive to K as long as its value is in the range of 80 to 200 (for a time horizon of 100000).

8 CONCLUSION

In this paper, we studied the optimal control problem in stochastic bipartite queueing networks, where we developed an admissible policy with low regret compared to the optimal dynamic policy. It is a first class of problems that focus on the challenges of combining learning and network control, where the learning aspect and the network control aspect are not separate and must be co-designed. We first showed that the expected utility of the optimal dynamic policy is upper bounded by T times the solution to a static linear program, where the optimization variables correspond to rates of jobs from dispatchers to servers and the feasibility region is parameterized by arrival rates and service rates. We next proposed the CB-JSQ-K policy for the optimal control problem that is an integration of an adapted version of the Confidence-Ball algorithm (learning algorithm) and the Dual-Level JSQ-K policy (control policy). The Dual-Level JSQ-K policy relies on the spanning forest structure induced by the extreme points of the static linear program while the Confidence-Ball algorithm seeks to learn the optimal extreme point. We proved that the CB-JSQ-K policy achieves logarithmic regret, which is superior to techniques in previous works that could only achieve $\Omega(\sqrt{T})$-regret.

There are several future directions. First, it would be interesting to consider utility functions that depend on the waiting time of the jobs instead of only the dispatcher and the server. The second direction involves the lower bound of the optimal routing problem Since the optimal control problem can be considered as a generalization of the multi-armed bandits problem, following from the lower bound of multi-armed bandits [11], a regret lower bound of $\Omega(\log T)$ also holds for the optimal routing problem. An important open problem is thus, whether stronger lower bounds (e.g. poly-logarithmic to T) hold for the optimal routing problem.

ACKNOWLEDGMENTS

This work was funded by Office of Naval Research (ONR) grant award N00014-20-1-2119, and by NSF grants CNS-2148128 and CNS-2148183.

REFERENCES

[1] J. Tsitsiklis and K. Xu. "Queueing system topologies with limited flexibility." in *Proceedings of the ACM SIGMETRICS*, pp: 167-178. 2013.
[2] W. Weng, X. Zhou, and R. Srikant. "Optimal load balancing in bipartite graphs." arXiv preprint arXiv:2008.08830 (2020).
[3] S. Banerjee, Y. Kanoria, and P. Qian. "State dependent control of closed queueing networks." in *ACM SIGMETRICS Performance Evaluation Review*, Vol. 46, No. 1, pp: 2-4, 2018.
[4] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy. "Optimal power allocation in server farms." in *ACM SIGMETRICS Performance Evaluation Review*, Vol. 37, No. 1, pp: 157-168, 2009.
[5] H. Yu, M. Neely, and X. Wei. "Online convex optimization with stochastic constraints." in *Advances in Neural Information Processing Systems*, 2017.
[6] S. Shalev-Shwartz, "Online learning and online convex optimization." in *Foundations and Trends in Machine Learning*, Vol. 4, No. 2, pp: 107-194, 2012.
[7] A. Agarwal, O. Dekel, and L. Xiao. "Optimal Algorithms for Online Convex Optimization with Multi-Point Bandit Feedback." in *Conference on Learning Theory*, pp: 28-40, 2010.
[8] T. Chen and G. B. Giannakis. "Bandit convex optimization for scalable and dynamic IoT management." in *IEEE Internet of Things Journal*, Vol. 6, No. 1, pp: 1276-1286, 2018.
[9] A. Agarwal, D. P. Foster, D. Hsu, S. M. Kakade, and A. Rakhlin. "Stochastic convex optimization with bandit feedback." in *SIAM Journal on Optimization*, Vol. 23, No. 1, pp: 213-240, 2013.
[10] J. Abernethy, E. Hazan, and A. Rakhlin. "Competing in the dark: An efficient algorithm for bandit linear optimization." 2009.
[11] S. Bubeck and N. Cesa-Bianchi, "Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems." in *Machine Learning*, Vol. 5, No. 1, pp: 1-122, 2012.
[12] J. He, D. Zhou, and Q. Gu. "Logarithmic regret for reinforcement learning with linear function approximation." in *International Conference on Machine Learning*, pp: 4171-4180, 2021.
[13] L. Zheng and L. Ratliff. "Constrained upper confidence reinforcement learning." in *Learning for Dynamics and Control*, pp: 620-629, 2020.
[14] I. Osband and B. Van Roy. "Near-optimal reinforcement learning in factored mdps." in *Advances in Neural Information Processing Systems*, 2014.
[15] M. Azar, I. Osband, and R. Munos. "Minimax regret bounds for reinforcement learning." in *International Conference on Machine Learning*, pp: 263-272, 2017.
[16] L. Yang and M. Wang. "Reinforcement learning in feature space: Matrix bandit, kernels, and regret bound." in *International Conference on Machine Learning*, pp: 10746-10756, 2020.
[17] V. Dani, T. Hayes, and S. Kakade. "Stochastic linear optimization under bandit feedback.", in *Conference on Learning Theory*, 2008.
[18] R. Singh and A. Stolyar. "Maxweight scheduling: Asymptotic behavior of unscaled queue-differentials in heavy traffic." in *Proceedings of ACM SIGMETRICS*, pp: 431-432, 2015.
[19] M. Neely. "Stochastic network optimization with application to communication and queueing systems." in *Synthesis Lectures on Communication Networks*, Vol. 3, No. 1, pp: 1-211, 2010.
[20] X. Lin, N. B. Shroff, and R. Srikant. "A tutorial on cross-layer optimization in wireless networks." in *IEEE Journal on Selected areas in Communications*, Vol. 24, No. 8, pp: 1452-1463, 2006.
[21] T. Stahlbuhk, B. Shrader, and E. Modiano. "Learning algorithms for minimizing queue length regret." in *IEEE Transactions on Information Theory*, Vol. 67, No. 3, pp: 1759-1781, 2021.
[22] S. Krishnasamy, R. Sen, R. Johari, and S. Shakkottai. "Learning unknown service rates in queues: A multiarmed bandit approach." in *Operations Research*, Vol. 69, No. 1, pp: 315-330, 2021.
[23] X. Fu, and E. Modiano. "Learning-NUM: Network Utility Maximization with Unknown Utility Functions and Queueing Delay." in *Proceedings of ACM Mobihoc*, pp. 21-30. 2021.
[24] X. Fu, and E. Modiano. "Elastic job scheduling with unknown utility functions." in *Performance Evaluation*, 2021.
[25] A. Vera and S. Banerjee. "The bayesian prophet: A low-regret framework for online decision making." in *Management Science*, Vol. 67, No. 3, pp: 1368-1391, 2021
[26] X. Tan, B. Sun, A. Leon-Garcia, Y. Wu, and D. Tsang. "Mechanism design for online resource allocation: A unified approach." in *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, Vol. 4, No. 2, pp: 1-46, 2020.

[27] Agrawal, Shipra, Zizhuo Wang, and Yinyu Ye. "A dynamic near-optimal algorithm for online linear programming." Operations Research 62, no. 4 (2014): 876-890.
[28] M. Neely. "Super-fast delay tradeoffs for utility optimal fair scheduling in wireless networks." in *IEEE Journal on Selected Areas in Communications*, Vol. 24, No. 8, pp: 1489-1501, 2006.
[29] D. Bertsimas and J. Tsitsiklis. "Introduction to linear optimization." Vol. 6. Belmont, MA: Athena Scientific, 1997.
[30] V. Pesic and R. Williams. "Dynamic scheduling for parallel server systems in heavy traffic: Graphical structure, decoupled workload matrix and some sufficient conditions for solvability of the Brownian control problem." in *Stochastic Systems*, Vol. 6, No. 1, pp: 26-89, 2016.
[31] J. M. Harrison and M. J. López. "Heavy traffic resource pooling in parallel-server systems." in *Queueing systems*, Vol. 33, No. 4, pp: 339-368, 1999.
[32] E. Hazan. "Introduction to online convex optimization." in *Foundations and Trends in Optimization*, Vol. 2, No. 3-4, pp: 157-325, 2016.

A PROOF OF PROPOSITION 2

Proof. The proof consists of three steps. We first show that the set of essential links forms a forest in the bipartite network, and each node in $\mathcal{G}$ is either connected to an essential link, or is an idle server, which implies that the essential links and idle servers form a spanning forest. Second, we prove that each tree in the spanning forest contains at most one slack server. Finally, we argue that for a non-degenerate extreme point, each tree in the forest contains exactly one slack server. The proof relies on the connection between the optimization problem $\mathcal{P}$ and the network flows in an extended network $\tilde{\mathcal{G}}$ of $\mathcal{G}$, which is defined as follows. An example of the extended network is shown in **Figure 6**.

Definition 2 (Extended Network). *The extended network is a flow network $\tilde{\mathcal{G}}$ with node set $\{\tilde{s}\} \cup \{\tilde{d}\} \cup \mathcal{U} \cup \mathcal{S}$. It link set consists of the links of $\mathcal{G}$, one link from $\tilde{s}$ to each $u_n \in \mathcal{U}$ and one link from each $s_m \in \mathcal{S}$ to $\tilde{d}$. The capacity of link $(\tilde{s}, u_n)$ is equal to λ_n, the capacity of link $(s_m, \tilde{d})$ is equal to μ_m, and the capacities of links between $\mathcal{U}$ and $\mathcal{S}$ are infinity.*

Consider the flow polytope formed by the set of $\tilde{s}$-$\tilde{d}$ flows $\mathcal{F}$ of value $\sum_n \lambda_n$ on $\tilde{G}$. Standard results in network flow [29] show that the flow polytope is equivalent to the feasibility region of $\mathcal{P}$, with the equivalence manifested by the flow value between (u_n, s_m) in $\mathcal{F}$ corresponding to the value of variable r_{nm} in $\mathcal{P}$. It follows that the extreme points of the flow polytope are equivalent to the extreme points of $\mathcal{P}$. For a $\tilde{s}$-$\tilde{d}$ flow in $\tilde{\mathcal{G}}$, we say a link is unsaturated if the flow value of the link is smaller than its capacity. The following lemma characterizes the structural property of extreme points in the flow polytope. It is also a standard result in the network flow literature and can be found in [29].

Lemma 1. *Under an extreme points, unsaturated links with positive flows do not form an undirected cycle.*

Based on Definition 2 and Lemma 1, we are ready to carry out the three steps in proving the proposition. For the first step, note that as the capacities of the links between $\mathcal{U}$ and $\mathcal{S}$ are infinity, those links are unsaturated under any flow. It follows that every essential link (in $\mathcal{G}$) corresponds to an unsaturated link with positive flow in the extended network $\tilde{\mathcal{G}}$. If the essential links do not from a forest (i.e., it contains a cycle), then there will be a cycle of unsaturated links with positive flows in the extended network, which contradicts the condition that $\{r_{nm}, y_m\}$ is an extreme point by Lemma 1. Furthermore, for each dispatcher u_n, as $\sum_{m \in \mathcal{S}_{u_n}} r_{nm} = \lambda_n > 0$, there must exist an $m \in \mathcal{S}_{u_n}$ with $r_{nm} > 0$. Thus, each dispatcher is connected to at least one essential link. For each server s_m, note that if s_m is not connected to any essential link, then we have $y_m = \mu_m > 0$ and s_m is an idle server. Therefore, the essential links and idle servers form a spanning forest of $\mathcal{G}$, with each tree in the forest is either a tree that contains essential links, servers and dispatchers, or a trivial tree that only contains an idle server.

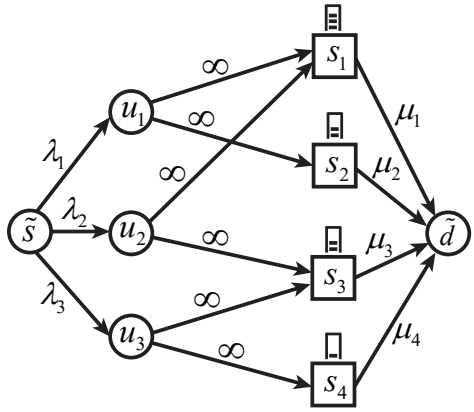

Fig. 6. An example of the extended network.

For the second step, consider an arbitrary tree in the forest induced by the extreme point. Suppose for the sake of contradiction, there exist two slack servers $s_m, s_{m'}$ in the tree. Then, the links $(s_m, \tilde{d})$ and $(s_{m'}, \tilde{d})$ are unsaturated with positive flows. Furthermore, since s_m and $s_{m'}$ are in the same tree, there exists a path of essential links connecting s_m and $s_{m'}$. Therefore, in the extended network $\tilde{G}$, the path together with links $(s_m, \tilde{d})$ and $(s_{m'}, \tilde{d})$ form a cycle of unsaturated links, which contradicts Lemma 1. Hence, there is at most one slack server in each tree.

Finally, if the extreme point is non-degenerate, then all variables in the basis must be strictly positive. Note that there are $N + M$ constraints in $\mathcal{P}$ (not including the non-negativity constraints). Hence, there are $N + M$ variables in the basis. Suppose that under the extreme point, there are k trees in the spanning forest. Then, as the spanning forest has $N + M$ nodes, it must have $N + M - k$ edges, which correspond to $N + M - k$ essential links. Since each essential link corresponds a basis variable, and as the link is non-degenerate, links with zero flow are not in the basis, it follows that there are k variables $y_m > 0$ in the basis that correspond to k slack servers. Since in the second step, we have shown that each tree contains at most one slack server, we thus have each tree contains exactly one slack server when the extreme point is non-degenerate. □

B REGRET LOWER BOUND OF STATIC POLICIES

In this section, we establish a lower bound on the regret of static policies. We formally define static policies as ones under which the numbers of jobs sent from each dispatcher u_n to server s_m, i.e., $a_{nm}(1), \ldots, a_{nm}(T)$ at time $t = 0, \ldots, T$ are independent random variables with the same mean. Note that we only require independence of decisions corresponding to each dispatcher-server pair across time, but do not ask for independence of $a_{nm}(t)$'s across different dispatcher-server pairs for the same t.[6] Also, the requirement is with respect to the routing policy at the dispatchers while the servers can employ an arbitrary scheduling policy. The lower bound is summarized as follow.

Proposition 4. *There exist instances of the optimal control problem in which any static policy has* $\Omega(\sqrt{T})$*-regret.*

Proof. We consider an instance with one dispatcher ($N = 1$) and two servers ($M = 2$). Server s_1 has an integer service rate μ_1 with its offered service $c_1(t)$ being an integer chosen from $\{\mu_1 - 1, \mu_1 + 1\}$

[6] In fact, they are often not independent as they have to satisfy $\sum_{m=1}^{M} a_{nm}(t) = a_n(t)$.

uniformly at random. Server s_2 has an integer service rate μ_2 with its offered service $c_2(t)$ being an integer chosen from $\{\mu_2 - 1, \mu_2 + 1\}$ uniformly at random. The underlying utilities $v_{11} = v_{12} + 1$. The arrival process is deterministic, with $a_1(t) = \lambda$ for each t. Note that we explicitly specify the distributions of the arrival and service processes only for concreteness. It should be clear from the proof that the result holds for a wide range of instances, not restricted to $M = 2$ or the distributions assumed here.

It is easy to see that the optimal solution to the linear program $\mathcal{P}$ is $r_{11}^* = \mu_1$ and $r_{12}^* = \lambda - \mu_1$. Hence, by Proposition 1,

$$U(\pi^*) \le \sum_{t=0}^{T} \mu_1 v_1 + \left[\sum_{t=0}^{T} \lambda - \sum_{t=0}^{T} \mu_1\right] v_2. \tag{5}$$

We consider an arbitrary static routing policy π that sends $a_{11}(t) = \lambda_1, t = 0, \ldots, T$ jobs to s_1 and $a_{12}(t) = \lambda_2, t = 0, \ldots, T$ jobs to s_2. We will show that the gap between (5) and $U(\pi)$ is $\Omega(\sqrt{T})$. Although (5) upper-bounds and may not be equal to $U(\pi^*)$, as the CB-JSQ-K policy achieves logarithmic regret, having a $\Omega(\sqrt{T})$ gap with respect to (5) implies a gap of the same order with respect to CB-JSQ-K, from which it will follow that any static policy has $\Omega(\sqrt{T})$-regret.

For the static policy π, by definition, we have

$$U(\pi) \le \left[\sum_{t=0}^{T} \lambda_1 - \mathbb{E}[Q_1(T)]\right] v_1 + \sum_{t=0}^{T} (\lambda - \lambda_1) v_2. \tag{6}$$

It follows that the gap between the right-hand-side of (5) and $U(\pi)$ is at least

$$\left[\sum_{t=0}^{T} \mu_1 - \sum_{t=0}^{T} \lambda_1 + \mathbb{E}[Q_1(T)]\right] v_1 + \sum_{t=0}^{T} (\mu_1 - \lambda_1) v_2. \tag{7}$$

Recall the evolution of Q_1 as $Q_1(t+1) = [Q_1(t) + a_{11}(t) - c_1(t)]^+ \ge Q_1(t) + a_{11}(t) - c_1(t)$. Let $\tilde{Q}_1(T) = \sum_{t=0}^{T} [a_{11}(t) - c_1(t)]$. It follows that $\mathbb{E}[Q_1(T)] \ge \mathbb{P}[\tilde{Q}_1(T) \ge 0] \cdot \mathbb{E}[\tilde{Q}_1(T) \mid \tilde{Q}_1(T) \ge 0]$. For any t, $a_1(t) - c_1(t)$ takes value in $\{\lambda_1 - \mu_1 - 1, \lambda_1 - \mu_1 + 1\}$ uniformly at random. It has the same distribution as $\lambda_1 - \mu_1 - 1 + 2 \cdot b(t)$ where $b(t)$ is a Bernoulli random variable that takes value 0 or 1 with equal probability.[7] Therefore, $\tilde{Q}_1(T)$ has the same distribution as $\sum_{t=0}^{T} (\lambda_1 - \mu_1) + 2 \cdot B(T+1, 1/2)$, where $B(T+1, 1/2)$ is a binomial random variable with parameters $T+1$, $1/2$. Let $\delta = \lambda_1 - \mu_1$. It follows that $\mathbb{E}[\tilde{Q}_1(T)] = \sum_{t=0}^{T} \delta$ and as $a_1(t)$, $c_1(t)$ are independent across time, the variance of $\tilde{Q}_1(T)$ is $T+1$. It follows that if $\epsilon \ge 0$ or $|\epsilon| = \Omega(1/\sqrt{T})$, we have

$$\mathbb{E}[Q_1(T)] \ge \mathbb{P}[\tilde{Q}_1(T) \ge 0] \cdot \mathbb{E}[\tilde{Q}_1(T) \mid \tilde{Q}_1(T) \ge 0] \ge \Omega(T\epsilon + \sqrt{T}) = \Omega(\sqrt{T}).$$

If $\epsilon < 0$ and $|\epsilon| = o(1/\sqrt{T})$, as still $\mathbb{E}[Q_1(T)] \ge 0$, we have

$$(7) \ge -\sum_{t=0}^{T} \epsilon = T|\epsilon| = \Omega(\sqrt{T}).$$

Therefore, in either case, the regret of a generic static policy π is $\Omega(\sqrt{T})$, which concludes the proof. □

[7] Equivalently, $a_{11}(t) - c_1(t) = \lambda_1 - \mu_1 + r(t)$ where $r(t)$ is a Rademacher random variable that takes value -1 or 1 with equal probability. Note that here the distribution of $a_{11}(t) - c_1(t)$ does not matter as long as its variance is a positive constant.

C PROOF OF THEOREM 1

Proof. In this section, we give the details of the proof of Theorem 1. Recall that for each dispatcher u_n (of priority level $H-h$) with primary servers $m_1,\dots,m_L$ and secondary server m_0, we will establish the following two claims:

Claim (1.1): Starting from $t = C_1 h \ln T$, $\sum_{t=C_1 h\ln T}^{T} \mathbb{P}[\exists i = 1,\dots,L, Q^h_{m_i}(t)+Q^l_{m_i}(t) \le (1-\epsilon_h)K] = O(\frac{1}{T})$.

Claim (1.2): The expected total completed service from u_n at each of the primary server m is close to $T\cdot r_{nm}$. For each $i=1,\dots,L$, $Tr_{nm_i} - O(\log T) \le \sum_{t=1}^{T}\mathbb{E}[\tilde{c}_{nm_i}(t)] \le Tr_{nm_i} + O(\log T)$.

For each server s_m (of priority level $H-h$) with primary dispatchers $n_1,\dots,n_L$ and secondary dispatcher n_0, we will establish the following two claims

Claim (2.1): Starting from $t = C_1 h \ln T$, $\sum_{t=C_1 h\ln T}^{T} \mathbb{P}[Q^h_m(t) \ge \epsilon_h K] = O(\frac{1}{T})$.

Claim (2.2): The expected total completed service at s_m from each of its primary dispatcher u_n is close to $T\cdot x_{nm}$. For each $i=1,\dots,L$, $Tr_{n_i m} - O(\log T) \le \sum_{t=1}^{T}\mathbb{E}[\tilde{c}_{n_i m}(t)] \le Tr_{n_i m} + O(\log T)$.

Combining the claims (1.2) and (2.2) of all the dispatchers and servers will conclude the proof of the theorem.

Base Step: The base step starts from the nodes of priority level $H-1$, since the nodes of priority level H are leaf nodes (with no children), of which the claims are trivial. We begin by the following fact, due to the arrivals and offered services being upper bounded by C.

Fact: Under JSQ-K, for any server s_m, $Q^h_m(t) \le K + C$ and $Q^l_m(t) \le K + C$ with probability 1.

Base Step – Case 1: We first consider the case where a node of priority level $H-1$ is a dispatcher u_n and prove the claims (1.1) and (1.2) with respect to the node. Note that its primary servers $m_1,\dots,m_L$ do not have primary dispatchers and are only connected to u_n as their secondary dispatcher in the tree.

Claim (1.1): Consider the potential function $Z_1(t) := L\cdot(K+C) - \sum_{i=1}^{L} Q^l_i(t)$ (we will write $Q^l_i(t)$ for $Q^l_{m_i}(t)$ for simplicity of notation). As $Q^l_i(t) \le K+C$ for all $i=1,\dots,L$, we have $\{\exists i = 1,\dots,L, Q^l_i(t) \le (1-\epsilon_1)K\} \subseteq \{\sum_{i=1}^{L} Q^l_i(t) \le (L-\epsilon_1)K + LC\}$, which implies that

$$\mathbb{P}\{\exists i = 1,\dots,L, Q^l_i(t) \le (1-\epsilon_1)K\} \le \mathbb{P}\left\{\sum_{i=1}^{L} Q^l_i(t) \le (L-\epsilon_1)K + LC\right\} = \mathbb{P}\{Z_1(t) \ge \epsilon_1 K\}. \quad (8)$$

Therefore, to prove Claim (1.1), we bound the probability that $Z_1(t) \ge \epsilon_1 K$.

Note that if there exists a primary server with queue length no larger than K, the incoming jobs of u_n will be sent to one of the primary servers. Also, we have $\{Z_1(t) \ge LC\} = \{\sum_{i=1}^{L} Q^l_i(t) \le L\cdot K\} \subseteq \{\exists i \in 1,\dots,L, Q^l_i(t) \le K\}$. It follows that

$$\mathbb{E}[Z_1(t+1) - Z_1(t) \mid Z_1(t) \ge LC] \le \mathbb{E}[-a_n(t) + \sum_{i=1}^{L} c_i(t) \mid Z_1(t) \ge LC] = -\lambda_n + \sum_{i=1}^{L}\mu_i = -r_{nm_0} \le -\Delta_2, \quad (9)$$

where the last inequality follows from the non-degeneracy of the extreme point.

Therefore, $Z_1(t)$ has conditional negative drift when $Z_1(t)$ is larger than $L\cdot C$. On the other hand, since the arrivals and offered services are bounded by C, $|Z_1(t+1) - Z_1(t)| \le LC$ with probability 1. Hence, we have

Let $\eta_1(t) = Z_1(t+1) - Z_1(t)$. We have for constant $\gamma = \frac{1}{4LC}$,

$$e^{\gamma Z_1(t+1)} = e^{\gamma Z_1(t)}\cdot e^{\gamma\eta_1(t)} \le e^{\gamma Z_1(t)}\cdot[1 + \gamma\eta_1(t) + 2\gamma^2\eta_1^2(t)].$$

It follows that

$$\mathbb{E}[e^{\gamma Z_1(t+1)} \mid Z_1(t) < LC] \le e^{\gamma Z_1(t)} \cdot e^{\gamma LC} \tag{10}$$

$$\mathbb{E}[e^{\gamma Z_1(t+1)} \mid Z_1(t) \ge LC] \le e^{\gamma Z_1(t)} \cdot \mathbb{E}[1 + \gamma\eta_1(t) - 2\gamma^2\eta_1^2(t)] \le e^{\gamma Z_1(t)} \cdot \left[1 - \frac{\Delta_2}{2}\gamma\right], \tag{11}$$

where the last inequality follows from $\mathbb{E}[\gamma\eta(t) \mid Z_1(t) \ge LC] = -\Delta_2\gamma$ and $2\gamma^2\eta_1^2(t) \le \frac{r|\eta_1(t)|}{2}$ for $\gamma = \frac{1}{4LC}$.

It follows that

$$\begin{aligned}
&\mathbb{E}[e^{\gamma Z_1(t+1)}] \\
=&\mathbb{P}[Z_1(t) \ge LC] \cdot \mathbb{E}[e^{\gamma Z_1(t+1)} \mid Z_1(t) \ge LC] + \mathbb{P}[Z_1(t) < LC] \cdot \mathbb{E}[e^{\gamma Z_1(t+1)} \mid Z_1(t) < LC] && (12)\\
\le&e^{\gamma LC} \cdot \mathbb{E}[e^{\gamma Z_1(t)} \mid Z_1(t) < LC] \cdot \mathbb{P}[Z_1(t) < LC] + \left[1 - \frac{\Delta_2\gamma}{2}\right] \cdot \mathbb{E}[e^{\gamma Z_1(t)} \mid Z_1(t) \ge LC] \cdot \mathbb{P}[Z_1(t) \ge LC] && (13)\\
=&\left[1 - \frac{\Delta_2\gamma}{2}\right] \cdot \mathbb{E}[e^{\gamma Z_1(t)}] + \left[e^{\gamma LC} - \left(1 - \frac{\Delta_2\gamma}{2}\right)\right] \cdot \mathbb{E}[e^{\gamma Z_1(t)} \mid Z_1(t) \le LC] \cdot \mathbb{P}[Z_1(t) \le LC] && (14)\\
\le&\left[1 - \frac{\Delta_2\gamma}{2}\right] \cdot \mathbb{E}[e^{\gamma Z_1(t)}] + e^{2\gamma LC}. && (15)
\end{aligned}$$

Iterating over inequality (15) and noting that $Z_1(0) = L(K + C)$, we obtain

$$\mathbb{E}[e^{\gamma Z_1(t)}] \le \frac{2e^{2\gamma LC}}{\Delta_2\gamma} + \left[1 - \frac{\Delta_2\gamma}{2}\right]^t e^{\gamma L(K+C)}. \tag{16}$$

It follows that after $t \ge C_1 \ln T$, $\left[1 - \frac{\Delta_2\gamma}{2}\right]^t e^{\gamma L(K+C)} \le 1$. Therefore,

$$\begin{aligned}
\mathbb{P}[Z_1(t) > \epsilon_1 K] = \mathbb{P}[e^{\gamma Z_1(t)} \ge e^{\epsilon_1\gamma K}] &\le \frac{\mathbb{E}[e^{\gamma Z_1(t)}]}{e^{\epsilon_1 rK}} && (17)\\
&\le \frac{2e^{\gamma(2C-\epsilon_1 K)}}{\Delta_2\gamma} + \frac{1}{e^{\epsilon_1\gamma K}} \le \frac{2}{T^2}, && (18)
\end{aligned}$$

Thus, we have $\sum_{t=C_1\log T}^{T} \mathbb{P}[\exists i = 1, \ldots, L, Q_{m_i}^l(t) \le (1 - \epsilon_1)K] \le T \cdot \frac{2}{T^2} = O(\frac{1}{T})$, which implies Claim (1.1).

Claim (1.2): For Claim (1.2), recall that $c_m(\omega, t)$ is the offered service of server m at time t on the sample path ω. Note that by construction, $r_{nm_i} = \mu_i$. Hence, we first have for each $i = 1, \ldots, L$, $\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{nm_i}(t)] \le \sum_{t=1}^{T} \mathbb{E}[c_{nm_i}(t, \omega)] = \sum_{t=1}^{T} \mu_i = Tr_{nm_i}$. Next, as $\tilde{c}_{nm_i}(t) = c_{nm_i}(t)$ on sample paths where there is no idleness in the queue of server s_m, we have that $Q_i^l(t) \ge (1 - \epsilon_1) \cdot K > C$ implies $\tilde{c}_{nm_i}(t) = c_{nm_i}(t)$. It follows that

$$\begin{aligned}
\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{nm_i}(t)] &\ge \sum_{t=1}^{T} \mathbb{E}[c_{nm_i}(t)] - \sum_{t=1}^{T} C \cdot \mathbb{P}[Q_i^l(t) < (1 - \epsilon_1) \cdot K] \\
&\ge Tr_{nm_i} - \sum_{t=C_1\ln T}^{T} C \cdot \mathbb{P}[Q_i^l(t) < (1 - \epsilon_1) \cdot K] - C \cdot C_1 \ln T \ge Tr_{nm_i} - O(\log T),
\end{aligned}$$

where the last inequality holds because of Claim (1.1) we just proved. This concludes the proof for the first case of the base step.

Base Step – Case 2: We next consider the case where a node of priority level $H-1$ is a server s_m and prove the claims (2.1) and (2.2) with respect to the nodes. Note that here, the server's primary dispatchers $m_1, \ldots, m_L$ are only connected to s_m as their secondary server in the tree.

Claim (2.1): In this case, the high-priority queue length $Q_m^h(t)$ it self can be used as the potential function for our analysis. For simplicity of notation, we will write $Z_2(t) \coloneqq Q_m^h(t)$. When $Z_2(t) \geq C$, there will be no idleness for the high-priority queue at server s_m at time t. Therefore,

$$\mathbb{E}[Z_2(t+1) - Z_2(t) \mid Z_2(t) \geq C] \leq \mathbb{E}\left[\sum_{i=1}^{L} a_i(t) - c_m(t) \mid Z_2(t) \geq C\right] = \sum_{i=1}^{L} \lambda_i - \mu_m = -r_{n_0 m} \leq -\Delta_2, \tag{19}$$

where the last inequality is due to the non-degeneracy of the extreme point. Also, $|Z_2(t+1) - Z_2(t)| \leq LC$ with probability 1. Therefore, let $\eta_2(t) = Z_2(t+1) - Z_2(t)$, We have for constant $\gamma = \frac{1}{4LC}$,

$$e^{\gamma Z_2(t+1)} = e^{\gamma Z_2(t)} \cdot e^{\gamma \eta_2(t)} \leq e^{\gamma Z_2(t)} \cdot [1 + \gamma \eta_2(t) + 2\gamma^2 \eta_2^2(t)].$$

It follows similarly as in (10) and (11) that

$$\mathbb{E}[e^{\gamma Z_2(t+1)} \mid Z_2(t) < C] \leq e^{\gamma Z_2(t)} \cdot e^{\gamma LC} \tag{20}$$

$$\mathbb{E}[e^{\gamma Z_2(t+1)} \mid Z_2(t) \geq C] \leq e^{\gamma Z_2(t)} \cdot [1 + \gamma \eta_2(t) - 2\gamma^2 \eta_2^2(t)] \leq e^{\gamma Z_2(t)} \cdot \left[1 - \frac{\Delta_2}{2}\gamma\right]. \tag{21}$$

Thus, we have

$$\begin{aligned}
&\mathbb{E}[e^{\gamma Z_2(t+1)}] \\
=&\mathbb{P}[Z_2(t) \geq C] \cdot \mathbb{E}[e^{\gamma Z_2(t+1)} \mid Z_2(t) \geq C] + \mathbb{P}[Z_2(t) < C] \cdot \mathbb{E}[e^{\gamma Z_2(t+1)} \mid Z_1(t) < C] && (22)\\
\leq& e^{\gamma LC} \cdot \mathbb{E}[e^{\gamma Z_2(t)} \mid Z_2(t) < C] \cdot \mathbb{P}[Z_2(t) < C] + \left[1 - \frac{\Delta_2 \gamma}{2}\right] \cdot \mathbb{E}[e^{\gamma Z_2(t)} \mid Z_2(t) \geq C] \cdot \mathbb{P}[Z_2(t) \geq C] && (23)\\
=& \left[1 - \frac{\Delta_2 \gamma}{2}\right] \cdot \mathbb{E}[e^{\gamma Z_2(t)}] + \left[e^{\gamma LC} - \left(1 - \frac{\Delta_2 \gamma}{2}\right)\right] \cdot \mathbb{E}[e^{\gamma Z_2(t)} \mid Z_2(t) \leq C] \cdot \mathbb{P}[Z_2(t) \leq C] && (24)\\
\leq& \left[1 - \frac{\Delta_2 \gamma}{2}\right] \cdot \mathbb{E}[e^{\gamma Z_2(t)}] + e^{\gamma (L+1)C}. && (25)
\end{aligned}$$

Iterating over inequality (25) and noting that $Z_2(0) = 0$, we obtain

$$\mathbb{E}[e^{\gamma Z_2(t)}] \leq \frac{2e^{\gamma(L+1)C}}{\Delta_2 \gamma} + \left[1 - \frac{\Delta_2 \gamma}{2}\right]^t. \tag{26}$$

It follows that,

$$\begin{aligned}
\mathbb{P}[Z_2(t) > \epsilon_1 K] = \mathbb{P}[e^{\gamma Z_2(t)} \geq e^{\gamma \epsilon_1 K}] &\leq \frac{\mathbb{E}[e^{\gamma Z_2(t)}]}{e^{\gamma \epsilon_1 K}} && (27)\\
&\leq \frac{2e^{\gamma(2C - \epsilon_1 K)}}{\Delta_2 \gamma} + \frac{1}{e^{\gamma \epsilon_1 K}} \leq \frac{2}{T^2}, && (28)
\end{aligned}$$

Thus, we have $\sum_{t=1}^{T} \mathbb{P}[Q_m^h(t) > \epsilon_1 K] = O(1)$, which implies the claim.

Claim (2.2): By definition of our JSQ-K policy, $r_{n_i m} = \lambda_i$. Hence, we first have for each $i = 1, \ldots, L$, $\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{n_i m}(t)] \leq \sum_{t=1}^{T} \mathbb{E}[a_n(t)] = \sum_{t=1}^{T} \mu_i = T r_{n_i m}$. Next, note that the incoming jobs from

dispatcher u_i are either discarded, still in the queue, or completed by T, and job discarding can only happen when $Q_m^h(t) > K$. It follows that,

$$\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{n_i m}(t)] \geq \sum_{t=1}^{T} \mathbb{E}[a_n(t)] - C \cdot \sum_{t=1}^{T} \mathbb{P}[Q_m^h(t) > K] - \mathbb{E}[Q_m^h(T)] \geq T r_{n_i m} - O(\log T), \quad (29)$$

where the last inequality holds because of Claim (2.1) we just proved and that $Q_m^h(T)$ no larger than $K + C$ almost surely. The concludes the proof for the second case of the base step.

Induction Step: We now proceed to the induction step of the proof. Suppose that Claims (1.1), (1.2), (2.1), (2.2) hold for nodes (dispatchers, servers) of priority levels $H, H-1, \ldots, H-h+1$. We consider a node of priority level $H-h$.

Induction Step – Case 1: Consider a dispatcher node u_n of priority level $H-h$. It is connected to its primary servers $m_1, \ldots, m_L$ of priority level $H-h+1$, and its secondary server m_0 of priority level $H-h-1$.

Claim (1.1): We consider the sub-tree rooted at u_n. Denote the set of dispatchers in the sub-tree (including u_n) as $\mathcal{U}_n$ and the set of servers in the sub-tree as $\mathcal{M}_n$. Let $\mathcal{M}_j$ be the set of server nodes of priority level $H-h+j$ in the sub-tree. Note that $\mathcal{M}_1 = \{m_0, \ldots, m_L\}$. Let $\mathcal{J} := \{3, 5, \ldots, h-1 \text{ or } h\}$ be the values of j corresponding to the server nodes in the sub-tree that are not directly connected to u_n. Let $M_h = \sum_{j\in\mathcal{J}} |\mathcal{M}_j|$, i.e., the total number of servers in the sub-tree. We consider the potential function $Z_1(t) := (M_h + L) \cdot (K + C) - \sum_{j\in\mathcal{J}} \sum_{m\in\mathcal{M}_j} [Q_m^h(t) + Q_m^l(t)] - \sum_{i=1}^{L} [Q_i^h(t) + Q_i^l(t)]$.

Note that by the induction hypothesis on Claim (1.1), we have that for each $j \in \mathcal{J}$, $\sum_{t=C_1(h-1)\ln T}^{T} \mathbb{P}\{\exists m \in \mathcal{M}_j, Q_m^h(t) + Q_m^l(t) \leq (1-\epsilon_j)K\} = O(1/T)$. Furthermore, by the induction hypothesis on Claim 2.1, we have that for each $i \in \{1, \ldots, L\}$, $\sum_{t=C_1(h-1)\ln T}^{T} \mathbb{P}\{Q_i^h(t) \geq \epsilon_{h-1}K\} = O(1/T)$. We will focus on the sample paths where none of the above holds, which have a total probability mass of $1 - O(1/T)$. This does not affect our regret bound since sample paths of mass $O(1/T)$ can only contribute $O(1)$-regret. Furthermore, it also does not affect the conditional expectations since if we condition on an event with probability $1 - O(1/T)$, the expectation is going to change by at most $O(1/T)$. Let $t_0 = C_1(h-1)\ln T$. For $t \geq t_0$, we have the following observations:

- As $Q_i^h(t) \leq \epsilon_{h-1}K$ and $Q_i^l(t) \leq K+C$ for all $i = 1, \ldots, L$, we have $Q_i^h(t)+Q_i^l(t) \leq (1+\epsilon_{h-1})K+C$.
- It follows that

$$\left\{\exists i = 1, \ldots, L, Q_i^h(t) + Q_i^l(t) \leq (1-\epsilon_h)K\right\} \subseteq \left\{\sum_{i=1}^{L} [Q_i^h(t) + Q_i^l(t)] \leq (L-1)((1+\epsilon_{h-1})K + C) + (1-\epsilon_h)K\right\} \quad (30)$$

$$= \left\{\sum_{i=1}^{L} [Q_i^h(t) + Q_i^l(t)] \leq (L - (\epsilon_h - (L-1)\epsilon_{h-1}))K + (L-1)C\right\}. \quad (31)$$

- Since for each $j \in \mathcal{J}, m \in \mathcal{M}_j$, $Q_m^h(t) + Q_m^l(t) \leq (1+\epsilon_j)K + C$, we have

$$\left\{\sum_{i=1}^{L} [Q_i^h(t) + Q_i^l(t)] \leq (L - (\epsilon_h - (L-1)\epsilon_{h-1}))K + (L-1)C\right\} \quad (32)$$

$$\subseteq \left\{\sum_{j\in\mathcal{J}} \sum_{m\in\mathcal{M}_j} [Q_m^h(t) + Q_m^l(t)] + \sum_{i=1}^{L} [Q_i^h(t) + Q_i^l(t)]\right.$$

$$\leq \sum_{j\in\mathcal{J}}\sum_{m\in\mathcal{M}_j}((1+\epsilon_j)K+C)+(L-(\epsilon_h-(L-1)\epsilon_{h-1}))K+(L-1)C\Bigg\} \tag{33}$$

$$=\Bigg\{\sum_{j\in\mathcal{J}}\sum_{m\in\mathcal{M}_j}[Q_m^h(t)+Q_m^l(t)]+\sum_{i=1}^{L}[Q_i^h(t)+Q_i^l(t)]$$

$$\leq M_h(K+C)+\left(\sum_{j\in\mathcal{J}}|\mathcal{M}_j|\epsilon_j\right)K+(L-(\epsilon_h-(L-1)\epsilon_{h-1}))K+(L-1)C\Bigg\} \tag{34}$$

$$\subseteq\left\{Z_1(t)\geq\left(\epsilon_h-(L-1)\epsilon_{h-1}-\sum_{j\in\mathcal{J}}|\mathcal{M}_j|\epsilon_j\right)K\right\}. \tag{35}$$

Let α_1 be $\left(\epsilon_h-(L-1)\epsilon_{h-1}-\sum_{j\in\mathcal{J}}|\mathcal{M}_j|\epsilon_j\right)$. To prove the claim, it suffices to bound the probability of $Z_1(t)\geq\alpha_1 K$. On the other hand, note that when there exists $i=1,\ldots,L$ with $Q_i^l(t)\leq K$, the incoming jobs from u_n will be sent to one of the servers $m_1,\ldots,m_L$. In this regard, we have the following observations:

- $\left\{\exists i=1,\ldots,L,\ Q_i^l(t)\leq K\right\}\supseteq\left\{\sum_{i=1}^{L}Q_i^h(t)+Q_i^l(t)\leq LK\right\}$.
- Since for each $j\in\mathcal{J}, m\in\mathcal{M}_j, Q_m^h(t)+Q_m^l(t)>(1-\epsilon_j)K$, we have

$$\left\{\sum_{i=1}^{L}Q_i^h(t)+Q_i^l(t)\leq LK\right\} \tag{36}$$

$$\supseteq\left\{\sum_{j\in\mathcal{J}}\sum_{m\in\mathcal{M}_j}[Q_m^h(t)+Q_m^l(t)]+\sum_{i=1}^{L}[Q_i^h(t)+Q_i^l(t)]\leq\sum_{j\in\mathcal{J}}|\mathcal{M}_j|(1-\epsilon_j)K+LK\right\} \tag{37}$$

$$=\left\{Z_1(t)\geq\sum_{j\in\mathcal{J}}|\mathcal{M}_j|\epsilon_j K+(M_h+L)C\right\} \tag{38}$$

Let β_1 be $\sum_{j\in\mathcal{J}}|\mathcal{M}_j|\epsilon_j$. It follows that

$$\mathbb{E}\left[Z_1(t+1)-Z_1(t)\mid Z_1(t)\geq\beta_1K+(M_h+L)C\right] \tag{39}$$

$$\leq\mathbb{E}\left[\sum_{m\in\mathcal{M}_n}c_m(t)-\sum_{n'\in\mathcal{U}_n}a_{n'}(t)\mid Z_1(t)\geq\beta_1K+(M_h+L)C\right] \tag{40}$$

$$=\sum_{m\in\mathcal{M}_n}\mu_m-\sum_{n'\in\mathcal{U}_n}\lambda_{n'}=-r_{nm_0}\leq-\Delta_2. \tag{41}$$

Let $\eta_1(t)=|Z_1(t+1)-Z_1(t)|$. We have that $|\eta_1(t)|\leq 2(|\mathcal{M}_n|+|\mathcal{U}_n|)C:=G_1\cdot C$ with probability 1. It follows that,

We have for any constant $r=\frac{1}{4G_1C}$,

$$e^{\gamma Z_1(t+1)}=e^{\gamma Z_1(t)}\cdot e^{\gamma\eta_1(t)}\leq e^{\gamma Z_1(t)}\cdot[1+\gamma\eta_1(t)+2\gamma^2\eta_1^2(t)].$$

Similarly as in (10) and (11), it follows that

$$\mathbb{E}[e^{\gamma Z_1(t+1)} \mid Z_1(t) < \beta_1 K + (M_h + L)C] \le e^{\gamma Z_1(t)} \cdot e^{\gamma G_1 C} \tag{42}$$

$$\mathbb{E}[e^{\gamma Z_1(t+1)} \mid Z_1(t) \ge \beta_1 K + (M_h + L)C] \le e^{\gamma Z_1(t)} \cdot [1 + \gamma\eta_1(t) - 2\gamma^2\eta_1^2(t)] \le e^{\gamma Z_1(t)} \cdot \left[1 - \frac{\Delta_2}{2}\gamma\right], \tag{43}$$

Going through a similar derivation as (15), we obtain that

$$\mathbb{E}[e^{\gamma Z_1(t+1)}] \le \left[1 - \frac{\Delta_2\gamma}{2}\right] \cdot \mathbb{E}[e^{\gamma Z_1(t)}] + e^{\gamma(G_1C + \beta_1 K + (M_h+L)C)}. \tag{44}$$

Iterating over inequality (15) and noting that $Z_1(t_0) \le (M_h + L)(K + C)$, we obtain

$$\mathbb{E}[e^{\gamma Z_1(t)}] \le \frac{2e^{\gamma((G_1+M+L)C+\beta_1 K)}}{\Delta_2\gamma} + \left[1 - \frac{\Delta_2\gamma}{2}\right]^{t-t_0} e^{\gamma(M_h+L)(K+C)}. \tag{45}$$

It follows that after $t - t_0 \ge C_1 \ln T$, $\left[1 - \frac{\Delta_2\gamma}{2}\right]^t e^{\gamma(M_h+L)(K+C)} \le 1$, i.e., $t \ge C_1 h \ln T$,

$$\mathbb{P}[Z_1(t) > \alpha_1 K] = \mathbb{P}[e^{\gamma Z_1(t)} \ge e^{\alpha_1 K r}] \le \frac{\mathbb{E}[e^{\gamma Z_1(t)}]}{e^{\alpha_1 K r}} \tag{46}$$

$$\le \frac{2e^{\gamma((G_1+M+L)C-(\alpha_1-\beta_1)K)}}{\Delta_2\gamma} + \frac{1}{e^{\alpha_1 K r}}, \tag{47}$$

Observe that

$$\alpha_1 - \beta_1 = \left(\epsilon_h - (L-1)\epsilon_{h-1} - \sum_{j\in\mathcal{J}} |\mathcal{M}_j|\epsilon_j\right) - \left(\sum_{j\in\mathcal{J}} |\mathcal{M}_j|\epsilon_j\right)$$

$$= \epsilon_h - (L-1)\epsilon_{h-1} - 2\sum_{j\in\mathcal{J}} |\mathcal{M}_j|\epsilon_j \ge \frac{\epsilon_h}{4} = \frac{1}{4\cdot(4(M+N))^{H-h}}.$$

It follows from (47) that $\mathbb{P}[Z_1(t) \ge \alpha_1 K] \le \frac{2}{T^2}$ for $t \ge C_1 h \ln T$. Thus, we have $\sum_{t=C_1 h\ln T}^{T} \mathbb{P}[\exists i = 1, \ldots, L, Q^h_{m_i}(t) + Q^l_{m_i}(t) \le (1-\epsilon_h)K] = O(\frac{1}{T})$.

Claim (1.2): For each primary server $i = 1, \ldots, L$ of u_n, we denote its set of primary dispatchers as $\mathcal{U}_i$. The number of jobs from u_n completed at each s_i is equal to the difference between the total number of jobs completed at s_i and the number of jobs from $\mathcal{U}_i$ completed at s_i. To establish an upper bound on $\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{nm_i}(t)]$, we observe that

$$\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{nm_i}(t)] \le \sum_{t=1}^{T} \mathbb{E}\left[c_{m_i}(t) - \sum_{n'\in\mathcal{U}_i} \tilde{c}_{n'm_i}(t)\right]. \tag{48}$$

Since by induction hypothesis of Claim (1.2) for the dispatchers in $\mathcal{U}_i$, we have for each $n' \in \mathcal{U}_i$, $\mathbb{E}[\sum_{t=1}^{T} \tilde{c}_{n'm_i}(t)] \ge T x_{n'm_i} - O(\log T)$. Hence,

$$\sum_{t=1}^{T} \mathbb{E}\left[c_{m_i}(t) - \sum_{n'\in\mathcal{U}_i} \tilde{c}_{n'm_i}(t)\right] \le T \cdot \left(\lambda_i - \sum_{n'\in\mathcal{U}} x_{n'm_i}\right) + O(\log T) = T \cdot x_{nm_i} + O(\log T), \tag{49}$$

where the last equality follows from the constraint satisfied by the extreme point.

To establish a lower bound on $\sum_{t=1}^{T}\mathbb{E}[\tilde{c}_{nm_i}(t)]$, we observe that

$$\sum_{t=1}^{T}\mathbb{E}[\tilde{c}_{nm_i}(t)] \tag{50}$$

$$\geq \sum_{t=1}^{T}\mathbb{E}\left[c_{m_i}(t)-\sum_{n'\in\mathcal{U}_i}\tilde{c}_{n'm_i}(t)\right]-C\cdot\sum_{t=1}^{T}\mathbb{P}\{Q_i^h(t)+Q_i^l(t)\leq C\}, \tag{51}$$

as $C\cdot\sum_{t=1}^{T}\mathbb{P}\{Q_i^h(t)+Q_i^l(t)\leq C\}$ upper-bounds the total amount of wasted service at server m_i due to idleness. By the induction hypothesis for dispatchers in $\mathcal{U}_i$, we have for each $n'\in\mathcal{U}_i$, $\mathbb{E}[\sum_{t=1}^{T}\tilde{c}_{n'm_i}(t)]\leq Tr_{n'm_i}+O(\log T)$. Combining this with the Claim (1.1) for dispatcher u_n that we just proved, we have

$$\sum_{t=1}^{T}\mathbb{E}\left[c_{m_i}(t)-\sum_{n'\in\mathcal{U}_i}\tilde{c}_{n'm_i}(t)\right]-C\cdot\sum_{t=1}^{T}\mathbb{E}[\mathbb{P}\{Q_i^h(t)+Q_i^l(t)\leq C\}] \tag{52}$$

$$\geq T\cdot\left(\lambda_i-\sum_{n'\in\mathcal{U}}r_{n'm_i}\right)-O(\log T)-O(\log T)=T\cdot r_{nm_i}-O(\log T), \tag{53}$$

from which Claim (1.2) follows.

Induction Step – Case 2. Consider a server node s_m of priority level $H-h$. It is connected to its primary dispatchers $n_1,\ldots,n_L$ of priority level $H-h+1$, and its secondary dispatcher n_0 of priority level $H-h-1$.

Claim (2.1): We consider the sub-tree rooted at s_m. Denote the set of dispatchers in the sub-tree as $\mathcal{U}_m$ and the set of servers in the sub-tree (including s_m) as $\mathcal{S}_m$. Let $\mathcal{M}_j$ be the set of server nodes of priority level $H-h+j$ in the sub-tree. Let $\mathcal{J}:=\{2,4,\ldots,h-1 \text{ or } h\}$ be the values of j corresponding to the server nodes in the sub-tree that are not s_m. Let $M_h=\sum_{j\in\mathcal{J}}|\mathcal{M}_j|$, i.e., the total number of servers (excluding s_m) in the sub-tree. We consider the potential function $Z_2(t):=\sum_{j\in\mathcal{J}}\sum_{m'\in\mathcal{M}_j}[Q_{m'}^h(t)+Q_{m'}^l(t)]+Q_m^h(t)$.

Again, similar as in the induction step of Claim (1.1), we focus on the sample paths where the high probability events of the inductive hypothesis hold, which have a probability mass of $1-O(1/T)$. For $t\geq t_0=C-1(h-1)\ln T$, according to the induction hypothesis, we have the following observations:

- Using Claim (1.1) of the induction hypothesis, for each $j\in\mathcal{J}, m'\in\mathcal{M}_j, Q_{m'}^h(t)+Q_{m'}^l(t)>(1-\epsilon_j)K$, we have

$$\left\{Q_m^h(t)\geq\delta_hK\right\} \tag{54}$$

$$\subseteq\left\{\sum_{j\in\mathcal{J}}\sum_{m\in\mathcal{M}_j}[Q_m^h(t)+Q_m^l(t)]+Q_m^h(t)\geq\sum_{j\in\mathcal{J}}|\mathcal{M}_j|(1-\epsilon_j)K+\epsilon_hK\right\} \tag{55}$$

$$=\left\{Z_2(t)\geq\sum_{j\in\mathcal{J}}|\mathcal{M}_j|(1-\epsilon_j)K+\epsilon_hK\right\} \tag{56}$$

  Let $\alpha_2=\sum_{j\in\mathcal{J}}|\mathcal{M}_j|(1-\epsilon_j)+\epsilon_h$. To establish the claim, it suffices to bound the probability of $Z_2(t)\geq\alpha_2K$.
- Using Claim (2.1) of the induction hypothesis, and that $Q_{m'}^l(t)\leq K+C$ for all m', we have that for $m'\in\mathcal{M}_j$, $Q_{m'}^h(t)+Q_{m'}^l(t)\leq(1+\epsilon_j)K+C$.

- When $Q_m^h(t) \geq C$, since $Q_m^h(t)$ is the high-priority queue of s_m and the offered service $c_m(t) \leq C$, at time t, all of the offered service of s_m will be used on the queue $Q_m^h(t)$. Furthermore,

$$\{Q_m^h(t) \geq C\} \tag{57}$$

$$\supseteq \left\{ \sum_{j\in\mathcal{J}} \sum_{m\in\mathcal{M}_j} [Q_m^h(t) + Q_m^l(t)] + Q_m^h(t) \geq C + \sum_{j\in\mathcal{J}} \sum_{m\in\mathcal{M}_j} ((1+\epsilon_j)K + C) \right\} \tag{58}$$

$$= \left\{ Z_2(t) \geq \sum_{j\in\mathcal{J}} |\mathcal{M}_j|(1+\epsilon_j)K + (M_h+1)C \right\}. \tag{59}$$

Furthermore, after $t \geq t_0$, for each $j \in \mathcal{J}, m' \in \mathcal{M}_j$ in the tree, $Q_{m'}^h(t) + Q_{m'}^l(t) \geq (1-\epsilon_j)K > C$. Hence, there will be no idleness in server $s_{m'}$. Let $\beta_2 = \sum_{j\in\mathcal{J}} |\mathcal{M}_j|(1+\epsilon_j)$. We have

$$\mathbb{E}\left[Z_2(t+1) - Z_2(t) \mid Z_2(t) \geq \beta_2 K + (M_h+1)C\right] \tag{60}$$

$$\leq \mathbb{E}\left[\sum_{n\in\mathcal{U}_m} a_n(t) - \sum_{m'\in\mathcal{M}_m} c'_m(t) \mid Z_2(t) \geq \beta_2 K + (M_h+1)C \right] \tag{61}$$

$$= \sum_{m'\in\mathcal{M}_m} \lambda'_m - \sum_{n\in\mathcal{U}_m} \mu_n = -r_{n_0 m} \leq -\Delta_2, \tag{62}$$

Let $\eta_2(t) := Z_2(t+1) - Z_2(t)$, and note that $|\eta_2(t)| \leq (|\mathcal{M}_m| + |\mathcal{U}_n|)C := G_2 C$ with probability 1.

It follows that, have for any constant $r = \frac{1}{4G_2C}$,

$$e^{\gamma Z_2(t+1)} = e^{\gamma Z_2(t)} \cdot e^{\gamma \eta_2(t)} \leq e^{\gamma Z_2(t)} \cdot [1 + r\eta_2(t) + 2\gamma^2 \eta_2^2(t)].$$

Similarly as in (10) and (11), we have

$$\mathbb{E}[e^{\gamma Z_2(t+1)} \mid Z_2(t) < \beta_2 K + (M_h+1)C] \leq e^{\gamma Z_2(t)} \cdot e^{\gamma G_2 C} \tag{63}$$

$$\mathbb{E}[e^{\gamma Z_2(t+1)} \mid Z_2(t) \geq \beta_2 K + (M_h+1)C] \leq e^{\gamma Z_2(t)} \cdot [1 + r\eta_2(t) - 2\gamma^2\eta_2^2(t)] \leq e^{\gamma Z_2(t)} \cdot \left[1 - \frac{\Delta_2}{2}\gamma\right]. \tag{64}$$

Going through a similar derivation as in (15), we obtain that

$$\mathbb{E}[e^{\gamma Z_2(t+1)}] \leq \left[1 - \frac{\Delta_2\gamma}{2}\right] \cdot \mathbb{E}[e^{\gamma Z_2(t)}] + e^{\gamma((G_2+M_h+1)C+\beta_2)}. \tag{65}$$

Iterating over inequality (65) and noting that $Z_2(t_0) \leq (M_h+1)\cdot(K+C)$, we obtain

$$\mathbb{E}[e^{\gamma Z_1(t)}] \leq \frac{2e^{\gamma((G_1+M+L)C+\beta_1 K)}}{\Delta_2\gamma} + \left[1 - \frac{\Delta_2\gamma}{2}\right]^{t-t_0} e^{\gamma(M_h+L)(K+C)}. \tag{66}$$

It follows that after $t - t_0 \geq C_1 \ln T$, $\left[1 - \frac{\Delta_2\gamma}{2}\right]^t e^{\gamma(M_h+1)(K+C)} \leq 1$, i.e., $t \geq C_1 h \ln T$,

$$\mathbb{P}[Z_2(t) > \alpha_2 K] = \mathbb{P}[e^{\gamma Z_1(t)} \geq e^{\alpha_2 K\gamma}] \leq \frac{\mathbb{E}[e^{\gamma Z_1(t)}]}{e^{\alpha_2 K\gamma}} \tag{67}$$

$$\leq \frac{2e^{\gamma(G_2+M_h+1)C-(\alpha_2-\beta_2)K)}}{\Delta_2\gamma} + \frac{1}{e^{\alpha_2 Kr}}, \tag{68}$$

Observe that

$$\alpha_2 - \beta_2 = \left(\sum_{j\in\mathcal{J}} |\mathcal{M}_j|(1-\epsilon_j) + \epsilon_h\right) - \left(\sum_{j\in\mathcal{J}} |\mathcal{M}_j|(1+\epsilon_j)\right)$$
$$= \epsilon_h - 2\sum_{j\in\mathcal{J}} |\mathcal{M}_j|\epsilon_j \geq \frac{1}{4\epsilon_h} = \frac{1}{4\cdot(4(M+N))^{H-h}}.$$

It follows from (68) that $\mathbb{P}[Z_2(t) \geq \alpha_2 K] \leq \frac{2}{T^2}$ for $t \geq C_1 h \ln T$. Thus, we have $\sum_{t=C_1 h\log T}^{T} \mathbb{P}[Q_m^h(t) \geq \epsilon_h K] = O(\frac{1}{T})$.

Claim (2.2): For each $i = 1, \ldots, L$, we first note that the number of jobs from u_i completed at s_m is upper bounded by the total number of jobs sent to s_m from u_i, which is further upper bounded by the difference between the total incoming jobs from u_i and the number of jobs completed by the primary servers of u_i. More formally, let $\mathcal{M}_i$ be the set of primary servers of u_i, we have

$$\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{n_i m}(t)] \geq \sum_{t=1}^{T} \mathbb{E}[a_{n_i}(t)] - \sum_{t=1}^{T}\sum_{m'\in\mathcal{M}_i} \mathbb{E}[\tilde{c}_{n_i m'}(t)]. \tag{69}$$

Thus, invoking the induction hypothesis on Claim (1.2) of u_i, we obtain that

$$\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{n_i m}(t)] \geq T\cdot\left(\lambda_i - \sum_{m'\in\mathcal{M}_i} r_{n_i m'}\right) - O(\log T) = T r_{n_i m} - O(\log T). \tag{70}$$

On the other hand, the total number of jobs completed at s_m from u_i is lower bounded by the total number of incoming jobs from u_i minus the total number of jobs completed at the primary servers of u_i, the number of unfinished jobs from u_i (in the queues), and the total number of jobs discarded from u_i due to all of its primary servers and secondary server (s_m) having queue length larger than K. It follows that,

$$\sum_{t=1}^{T} \mathbb{E}[\tilde{c}_{n_i m}(t)] \tag{71}$$
$$\geq \sum_{t=1}^{T} \mathbb{E}[a_{n_i}(t)] - \sum_{t=1}^{T}\sum_{m'\in\mathcal{M}_i} \mathbb{E}[\tilde{c}_{n_i m'}(t)] - \sum_{m'\in\mathcal{M}_i} \mathbb{E}[Q_{m'}^l(T)] - \mathbb{E}[Q_m^h(T)] - C\cdot\sum_{t=1}^{T}\mathbb{P}\{Q_m^h(t) > K\} \tag{72}$$
$$\geq \left(\lambda_i - \sum_{m'\in\mathcal{M}_i} r_{n_i m'}\right) - O(\log T) - O(\log T) - O(\log T) = T r_{n_i m} - O(\log T), \tag{73}$$

where the last inequality follows from Claim (2.1) of s_m that we just proved and the induction hypothesis of Claim (1.2) on u_i. □

Remark: from the proof of Theorem 1 we see that to achieve a logarithmic utility gap, it suffices to set K as $8\cdot[4(M+N)]^H \log T$, where H is the maximum height of the trees in the forest induced by the extreme point. Therefore, the value of K need not depend on Δ_2 (although the upper bound on the utility gap is proportional to $\frac{1}{\Delta_2}$). Moreover, as we did not tighten the bound in terms of constant factors the constant factor in the value of K could actually be set as a much smaller value.

D PROOF OF THEOREM 2

Proof. Throughout the proof, we will refer to the events that happen with probability at least $1 - O(\frac{1}{T})$ as events "with high probability". Similarly as in Theorem 1, the threshold K is chosen to be $8(4(M+N))^H \log T$, where H upper-bounds the maximum height of the tree in the forest induced by any extreme point (e.g. $H = M + N$). Throughout the proof, we will make arguments about the extreme points of $\mathcal{P}$, which are most conveniently stated with respect to the standard form of $\mathcal{P}$. Recall that the standard form of $\mathcal{P}$ has a feasibility region of the form $D = \{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$, where $\boldsymbol{x} = \{r_{nm}, y_m\}$ and $\boldsymbol{A}$ represents the constraint matrix and $\boldsymbol{b}$ represents the constraint vector (formed by the arrival rates and service rates). We use $\boldsymbol{B} \subseteq \boldsymbol{A}$ to denote a generic basis (k columns of $\boldsymbol{A}$). We will use $\boldsymbol{x}$ or $\boldsymbol{r}$ (when we are only focusing on the $\{r_{nm}\}$ components) to denote a generic extreme point.

The key of the proof is to show the claim that with high probability, we will be able obtain at least one utility observation of (u_n, s_m)-job for each $r_{nm} > 0$. This, combined with Corollary 3 and Theorem 1 will lead to the proof of the theorem. We formally state the claim in the Lemma 2.

Lemma 2. *With probability* $1 - O(\frac{1}{T})$*, for each episode* $e \geq 2$*, at least one* (u_n, s_m)*-job is completed during episode* e *for every* $r^e_{nm} > 0$.

Proof of Lemma 2: First, we note that after $e \geq 2$, $t_e \geq \log^2 T$. It follows from Azuma-Hoeffding inequality that with high probability, $|\hat{\lambda}^t_n - \lambda_n| = O\left(\sqrt{\frac{1}{\log T}}\right)$, $|\hat{\mu}^t_m - \mu_m| = O\left(\sqrt{\frac{1}{\log T}}\right)$ for all $n, m, t \geq t_e$. Consider a generic episode $e \geq 2$ with the extreme point $\boldsymbol{r}^e$ of $\hat{D}_e$ used in the CB-JSQ-K policy.

Let $\boldsymbol{B}_e$ be the basis corresponding to $\boldsymbol{r}_e$. We will begin by showing that $\boldsymbol{B}_e$ is a feasible basis with respect to the true feasibility region D of $\mathcal{P}$. Indeed, let $\boldsymbol{b}_e$ be the constraint vector of $\hat{D}_e$ based on the empirical means of arrivals and services. Since $\boldsymbol{r}_e$ is feasible with respect to $\hat{D}_e$, we have $\boldsymbol{B}_e^{-1}\boldsymbol{b}_e \geq \boldsymbol{0}$. Recall that $\boldsymbol{b}$ is the constraint vector of the true feasibility region D. Since with high probability, $||\boldsymbol{b} - \boldsymbol{b}_e|| = O(\sqrt{\frac{1}{\log T}})$, for each component i, $(\boldsymbol{B}_e^{-1}\boldsymbol{b})_i \geq -\hat{C} \cdot \sqrt{\frac{1}{\log T}}$ for some constant $\hat{C}$. Suppose for the sake of contradiction that $\boldsymbol{B}_e$ is not feasible with respect to D, then by condition 2, there exists a component i such that $(\boldsymbol{B}_e^{-1}\boldsymbol{b})_i \leq -\Delta_2$, which leads to a contradiction. Therefore, $\boldsymbol{B}_e$ is a feasible basis with respect to D. It follows that all the drift inequalities (under the arrival rates $\{\lambda_n\}$ and service rates $\{\mu_m\}$) in the proof of Theorem 1 holds for the JSQ-K policy based on the forest $\mathcal{T}_e$ except possibly for the root servers (since the extreme point can be degenerate). Thus, after $O(\log T)$ time slots from the beginning of the episode, with high probability, the servers (except for the root servers) are never idle, and the dispatchers never discard incoming jobs (except possibly for the dispatchers directly connected to a root server). We will again focus on this set of sample paths, which as have been shown will not affect the regret analysis.

Now, we will show the claim that with high probability, at least one (u_n, s_m)-job is completed for every $r^e_{nm} > 0$ during episode e. First, we consider the case where u_n is a primary dispatcher of s_m, i.e., the jobs from u_n receives priority service at s_m in its (virtual) priority queue Q^h_m. Note that as $Q^h_m(t) \leq K + C = O(\log T)$, the queueing delay experienced by any (u_n, s_m)-job is in $O(\log T)$ with high probability. Furthermore, we show that within $O(\log T)$ time slots from t_e, dispatcher u_n will send at least one job to Q^h_m. This, combined with the previous argument will prove the claim. Indeed, let $m_1, \ldots, m_L$ be the primary servers of u_n. By the construction of JQS-K, dispatcher u_n will send incoming jobs to s_m when $Q^l_{m_i} > K$ for all $i = 1, \ldots, L$. Let $\mathcal{M}_n$ be the set of servers in the sub-tree of u_n. From previous discussion and using condition 2, we have that the function $\sum_{m' \in \mathcal{M}_n} Q^h_{m'}(t) + Q^l_{m'}(t)$ has positive drift of at least Δ_2 if there exists $i = 1, \ldots, L$ with $Q^l_{m_i}(t) \leq K$. Note that this still holds even when s_m is a root server. Hence, following a similar drift analysis in

the proof of Theorem 1, after at most $O(\log T)$ time slots, all queues $Q^l_{m_i}$ for $i = 1, \ldots, L$ will be greater than K and the u_n will send the incoming jobs to s_m.

Second, we consider the case where u_n is the secondary dispatcher of s_m, i.e., the jobs from u_n receives service at s_m in its (virtual) low-priority queue Q^l_m when Q^h_m is empty. We will show that with high probability, Q^h_m will be empty (and jobs in Q^l_m will be served) every $O(\log T)$ time slots. Indeed, consider the sub-tree at s_m and let $\mathcal{M}_m$ be the set of servers in the sub-tree (not including s_m). From previous discussion, we have that the function $Q^a_m(t) + \sum_{m' \in \mathcal{M}_m} Q^h_{m'}(t) + Q^l_{m'}(t)$ has a negative drift of at least Δ_2 when $Q^a_m(t)$ is not idle. Hence, following a similar drift analysis in the proof of Theorem 1, after at most $O(\log T)$ time slots, Q^h_m will be idle and Q^l_m will receive service. Since Q^l_m is bounded by $K + C = O(\log T)$, it will follow that the queueing delay experienced by any job in Q^l_m is in $O(\log^2 T)$. As s_m is a primary server of u_n, there will be at least one (u_n, s_m)-job in Q^l_m within $O(\log T)$ slots from t_e. Combining the above arguments, we prove that there is at least one (u_n, s_m)-job completed in the episode of length $W = \log^2 T \log\log T$ with high probability. □

From Lemma 2, we see that we can focus on the set of sample paths where the necessary utility observations are obtained for every episode $e \geq 2$. Let $\boldsymbol{r}^*_e$ be the optimal extreme point (with respect to the true utility vector $\boldsymbol{v}$) of $\hat{D}_e$. The analysis in [17] directly implies the following extension of Corollary 3 that with probability at least $1 - \frac{1}{T}$, $\boldsymbol{r}_e \neq \boldsymbol{r}^*_e$ for $O(\log^3 T)$ episodes. We will proceed to show in Lemma 3 that for $e \geq 2$, $\boldsymbol{r}^*_e$ induces the same forest as the optimal extreme point $\boldsymbol{r}^*$ of D.

Lemma 3. *For $e \geq 2$, the optimal extreme point of $\hat{D}_e$ induces the same forest as the optimal extreme point of D.*

Proof of Lemma 3: Let B^* be the basis of the optimal extreme point $\boldsymbol{r}^*$ of D. Since from the previous discussion, for $e \geq 2$, $||\boldsymbol{b} - \boldsymbol{b}_e|| = O(\sqrt{\frac{1}{\log T}})$ with high probability, we have that $\boldsymbol{B}^*$ is also a feasible basis for $\hat{D}_e$. Suppose for the sake of contradiction, the optimal extreme point $\boldsymbol{r}_e$ of $\hat{D}_e$ has a different basis $\boldsymbol{B}_e$. From the proof of Lemma 3, we see that $\boldsymbol{B}_e$ is also a feasible extreme point of D. Let $\tilde{\boldsymbol{r}} := \boldsymbol{B}_e^{-1}\boldsymbol{b}$ be the extreme point of D with respect to $\boldsymbol{B}_e$, and $\boldsymbol{r}^*_e := (\boldsymbol{B}^*)^{-1}\boldsymbol{b}_e$ be the extreme point of $\hat{D}_e$ corresponding to the basis $\boldsymbol{B}^*$. We have that $||\boldsymbol{r}^*_e - \boldsymbol{r}^*|| = O(\sqrt{\frac{1}{\log T}})$ and $||\boldsymbol{r}_e - \tilde{\boldsymbol{r}}|| = O(\sqrt{\frac{1}{\log T}})$. It follows that $|\boldsymbol{v} \cdot \boldsymbol{r}^* - \boldsymbol{v} \cdot \tilde{\boldsymbol{r}}| = O(\sqrt{\frac{1}{\log T}})$, which contradicts condition 1. Thus, we conclude the proof of the lemma.

From Lemma 3, we conclude that with high probability, there are at most $O(\log^3 T)$ episodes where $\mathcal{T}_e$ is not equal to the forest induced by the optimal extreme point $\boldsymbol{r}^*$. We divide the time horizon into periods, where each period is formed by consecutive episodes with the same forest. The regret of the CB-JSQ-K policy is the sum of the regret over each period. We will call a period/episode correct if the spanning forest of the period/episode coincides with the optimal, and a period/episode incorrect otherwise. Since there are at most $O(\log^3 T)$ episodes where $\mathcal{T}_e$ is not equal to the optimal forest, there are at most $O(\log^3 T)$ periods. The total length of incorrect period is upper bounded by the total number of incorrect episodes times the episode length, which is $O(\log^5 T \log\log T)$ time slots. Therefore, the regret incurred in incorrect period is in $O(\log^5 T \log\log T)$. Whenever the policy switches between periods, it discards all the jobs in the queue, which will in total incur $O(\log^4 T)$-regret as the total queue length is in $O(\log T)$. Finally, by Theorem 1, as the optimal extreme point is non-degenerate, the regret incurred in each correct period is $O(\log T)$. Therefore, in summary, the regret of the CB-JSQ-K policy is in $O(\log^5 T \log\log T)$.

Received August 2022; revised October 2022; accepted November 2022