The Value-of-Information in Matching with Queues

Longbo Huang
longbohuang@tsinghua.edu.cn
IIIS, Tsinghua University

***Abstract*—We consider the problem of *optimal matching with queues* in dynamic systems and investigate the value-of-information. In such systems, operators match tasks and resources stored in queues, with the objective of maximizing the system utility of the matching reward profile, minus the average matching cost. This problem appears in many practical systems and the main challenges are the no-underflow constraints, and the lack of matching-reward information and system dynamics statistics. We develop two online matching algorithms: Learning-aided Reward optimAl Matching (`LRAM`) and Dual-`LRAM` (`DRAM`) to effectively resolve both challenges. Both algorithms are equipped with a learning module for estimating the matching-reward information, while `DRAM` incorporates an additional module for learning the system dynamics. We show that both algorithms achieve an $O(\epsilon + \delta_r)$ close-to-optimal utility performance for any $\epsilon > 0$, while `DRAM` achieves a faster convergence speed and a better delay compared to `LRAM`, i.e., $O(\delta_\pi/\epsilon + \log(1/\epsilon)^2)$ delay and $O(\delta_\pi/\epsilon)$ convergence under `DRAM` compared to $O(1/\epsilon)$ delay and convergence under `LRAM` (δ_r and δ_π are maximum estimation errors for reward and system dynamics). Our results show that information of different system components can play very different roles in algorithm performance and provide a novel way for designing joint learning-control algorithms.**

I. Introduction

Matching is a fundamental problem that appears in resource allocation in various systems across different areas. For instance, network switch scheduling [1], online advertising [2], crowdsourcing [3], ride sharing [4], cloud computing [5], and inventory control [6]. As a result, efficient matching algorithms are of great importance to system control.

In this paper, we study the problem of optimal matching with queues in a dynamic environment with unknown matching reward statistics. Specifically, we consider a system consisting of a set of *task queues* and a set of *resource queues*, which store different types of workload and different types of resources that come into the system according to some random processes. At every time, the system operator decides how to match the resources to the pending workload. Each matching incurs a *cost* that depends on the resource allocated and random factors in the system, e.g., changing channel conditions in a downlink system, time-varying prices in inventory control, or fluctuating payment requirements in crowdsourcing. On the other hand, the matching also generates a *reward*, which is random with an unknown distribution determined by the amount of tasks resolved and the system condition. The objective is to design a matching strategy that carefully manages resources and tasks, so as to achieve optimal system utility, which is a function of the achieved reward profile, subject to the constraint that all tasks are fulfilled timely.

This is a general problem and models the aforementioned application scenarios. However, it is non-trivial to solve. First, the system utility is a function of the matching reward, which means that it is affected by when and how much resource is actually matched to the tasks and is only indirectly related to the traffic rates. This differs from traditional flow utility optimization problems [7], [8], and requires both careful admission control to avoid instability and appropriate matching to achieve good utility. Second, since each matching action is rewarded based on the actual amount of tasks resolved, the matching scheme must ensure that there are nonzero tasks and nonzero resources in the queues, i.e., *no-underflow*. This constraint is complex and is mostly tackled with dynamic programming, which can have high computational complexity. Third, the system is dynamic and the statistics of system conditions and reward functions are unknown beforehand. This requires that the matching scheme can efficiently learn the sufficient statistics of the randomness and adapt to the changing environment.

In addition to resolving the above challenges, we also try to investigate the *value-of-information* in such matching-with-queue systems, by explicitly considering the impact of information on algorithm performance. Existing works on stochastic system control either focus on systems with perfect a-prior information, e.g., [9], [10], or rely on stochastic approximation techniques that do not require such information, e.g., [11], [12]. While the proposed solutions are effective, they do not capture how information affect algorithm design and performance, and do not provide interfaces for integrating the fast-developing "data science" tools, e.g., data collecting methods and machine learning algorithms, [13], [14], into system control.

To provide a rigorous quantification of the value-of-information, we first introduce an abstract notion of a *learning module*, which represents a general information learning algorithm and features a learning accuracy level δ (maximum error), a learning time T_δ, and the probability of learning accuracy guarantee P_δ. We then design two online matching algorithms: Learning-aided Reward optimAl Matching (`LRAM`) and Dual-`LRAM` (`DRAM`). `LRAM` utilizes a single $(T_{\delta_r}, \delta_r, P_{\delta_r})$ learning module for estimating the reward statistics and achieves an $O(\epsilon + \delta_r)$ system utility, for any $\epsilon > 0$, while ensuring an $O(1/\epsilon)$ delay bound and an $O(1/\epsilon)$ algorithm con-

This paper was presented in part at the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), Hangzhou, China, June 2015.

This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61361136003, 61303195, Tsinghua Initiative Research Grant, Microsoft Research Asia Collaborative Research Award, and the China Youth 1000-talent Grant.

vergence time, defined to be the time taken for the algorithm to enter the optimal control state. DRAM incorporates an additional $(T_{\delta_\pi}, \delta_\pi, P_{\delta_\pi})$ learning module for the random system state distribution and guarantees a similar $O(\epsilon + \delta_r)$ system utility. Moreover, DRAM is able to achieve an $O(\delta_\pi/\epsilon + \log(1/\epsilon)^2)$ delay bound and an $O(\delta_\pi/\epsilon)$ algorithm convergence time, which can be significantly faster compared to LRAM. Our formulation and algorithms share similarity with those in the multi-armed bandit literature, e.g., [15], [16]. However, our results differ in that (i) queue underflow affects reward, and (ii) queueing delay is explicitly considered and analyzed.

Our results reveal an interesting fact that the reward information largely determines the utility performance, while the system dynamics information greatly affects delay and algorithm convergence. This indicates that *information of different system components can have different impacts on algorithm performance, and may require different learning power for achieving a desired goal*. We also explicitly quantify the values of different system information. Closest to our paper are recent works [17] and [18], which consider joint learning and control. However, [17] and [18] both assume prior knowledge of the action-reward functions and consider a specific learning approach, whereas our framework allows more general learning methods, and handles the no-underflow constraints and unknown rewards. Moreover, the analysis here is different from those in [17] and [18], due to the potential estimation errors in learning the reward functions. We summarize the main contributions as follows:

1) We propose a matching queueing system model, which models general resource-task matching problems in stochastic systems. To explicitly quantify the value of information in such systems, we introduce an abstract notion of a $(T_\delta, \delta, P_\delta)$-*learning module* that captures key characteristics of general learning algorithms and provides interfaces for bringing the information learning aspect into system control.
2) We design two learning-aided matching algorithms LRAM and DRAM. We show that with a single $(T_{\delta_r}, \delta_r, P_{\delta_r})$-module for learning reward statistics, LRAM achieves an $O(\epsilon+\delta_r)$ utility, while ensuring an $O(1/\epsilon)$ delay bound and an $O(1/\epsilon)$ algorithm convergence time. DRAM adopts an additional $(T_{\delta_\pi}, \delta_\pi, P_{\delta_\pi})$-module for learning the system state distribution, and guarantees a similar $O(\epsilon+\delta_r)$ system utility, while achieving an $O(\delta_\pi/\epsilon + \log(1/\epsilon)^2)$ delay and an $O(\delta_\pi/\epsilon)$ convergence time. We also construct two $(O(1/\epsilon^c), O(\epsilon^{c/2}), 1 - O(\epsilon^{\log(1/\epsilon)}))$ online learning modules based on sampling ($c > 0$). Combining them with DRAM, one achieves a fast $O(1/\epsilon^{1-c/2} + 1/\epsilon^c)$ convergence time with $c < 1$ (existing algorithms require $\Theta(1/\epsilon)$).
3) Our algorithm design approach provides a low-complexity way to tackle multiple simultaneous no-underflow constraints in systems and optimize utilities that are not purely defined on flow rates. The development of DRAM also demonstrates how general learning algorithms can be combined with queue-based control (stochastic approximation) to achieve superior delay and accelerate algorithm convergence speed.

The rest of the paper is organized as follows. We first list a few motivating examples in Section II. We then present the matching system model in Section III. The algorithm design approach and the two algorithms LRAM and DRAM are presented in Section IV. Analysis is carried out in Section VII and simulation results are presented in Section VIII. We then conclude the paper in Section IX.

II. Motivating Examples

Crowdsourcing: In a crowdsourcing application, e.g., crowdsourcing query search [3] or ride-sharing [4], tasks of different types (task) arrive at the server and are assigned to workers (resource). The workers then carry out the tasks. Depending on workers' qualifications, types of jobs, and the instantaneous system condition (state), e.g., whether a query requestor is in a hurry due to weather, requestors receive certain rewards, e.g., satisfaction, and workers receive payments. The objective of the system is to design a matching scheme, so as to maximize system utility, which is a function of the achieved requestor reward profile.

Energy Harvesting Networks: In an energy harvesting network, e.g., [19], [20], nodes are responsible for transmitting data (task) and can harvest energy (resource) from the environment. At every time, each node decides how much energy to allocate for transmission and determines traffic scheduling. Depending on the time-varying channel condition (state), amount of energy enables certain processing results. The objective is to design a joint energy management and scheduling algorithm, so as to maximize traffic utility and ensure that no energy outage happens.

Online Advertisement: In an online advertising system, [2], [21], advertisers deposit money (task) into their accounts at the advertising platform. Queries (resource) for different keywords arrive in the system and the server decides which advertiser's ads to show, based on their relevance to the keywords and the available budget of the advertisers. Depending on the chosen ad and the user's condition (state), e.g., location or mood, a business transaction may take place. The goal of the system is to design an ad matching scheme, so as to maximize system utility, which is a function of the average income profile from advertisers.[1]

Cloud Computing: In a cloud computing platform, e.g., [5], computing resources (resource), e.g., CPU and memory, are assigned to virtual machine instances (task) for processing arriving job requests. The quality of experience of a requestor depends on the job completion quality, which is affected by system conditions such as background task level (state) and user status. The objective here is to design a resource allocation policy, such that the overall quality of service is maximized.

In all these examples, the underlying problem is indeed matching with queues. Below, we present the general model.

III. System Model

We consider a discrete-time system shown in Fig. 1. In this system, there are two sets of queues, *task queues* and *resource*

[1]Here we view queries as opportunities (resource) for earning the deposit (task) from advertisers, e.g., through pay-per-click.

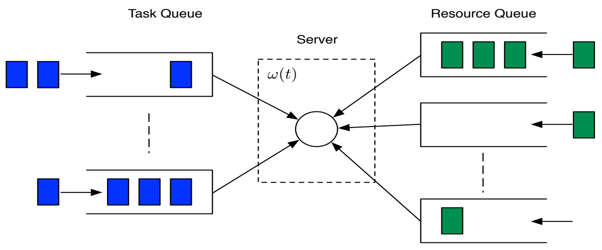


Fig. 1. The matching queueing system.

queues, and a central server (called operator below), which coordinates resource allocation and scheduling in the system. Time is divided into unit-size slots, i.e., $t \in \{0, 1, ...\}$.

A. Tasks and Resources

The task queues store jobs that come into the system and are waiting to be served by the server. We assume there are N types of tasks and denote the set of task queues by $\mathcal{Q} = \{\mathcal{Q}_1, ..., \mathcal{Q}_N\}$. We use $A_n(t)$ to denote the amount of new tasks arrivaling at $\mathcal{Q}_n$ at time t and assume that $0 \leq A_n(t) \leq A_{\max}$. We then define the arrival vector $\boldsymbol{A}(t) = (A_1(t), ..., A_N(t))$. In many systems, arrivals to the system may not always all be admitted due to congestion control, e.g., when all servers are busy. We model this by using $0 \leq R_n(t) \leq A_n(t)$ to denote the actual admitted traffic to $\mathcal{Q}_n$ at time t. We then use $Q_n(t)$ to denote the amount of tasks stored at $\mathcal{Q}_n$ at time t and denote $\boldsymbol{Q}(t) = (Q_1(t), ..., Q_N(t))$ the task queue vector.

The resource queues, on the other hand, hold the resources the system collects over time. There are M types of system resources and we denote the resource queues by $\mathcal{H} = \{\mathcal{H}_1, ..., \mathcal{H}_M\}$. We similarly let $e_m(t)$ be the amount of new resource arriving at $\mathcal{H}_m$ with $0 \leq e_m(t) \leq h_{\max}$. We also use $H_m(t)$ to denote the amount of resource m the system currently holds and denote $\boldsymbol{H}(t) = (H_1(t), ..., H_M(t))$ the resource queue vector.

In many systems, it is feasible (and sometimes necessary) to control the amount of resources in the system, e.g., to avoid too many workers waiting in crowdsourcing. We model this decision by using $h_m(t) \in [0, e_m(t)]$ to denote the actual amount of type m resource admitted. For now, it is also convenient to temporarily assume that the queues are all of unlimited sizes. We will later show that our algorithms ensure that finite buffer sizes are sufficient.

B. System State and Resource allocation

We assume that the system condition varies over time, e.g., channel conditions in a downlink system, or the expected happiness measures of human users in a crowdsourcing system. We call this condition the *system state* and model it by a random state variable $\omega(t)$. Note that $\omega(t)$ represents the *aggregate* system condition. For instance, in an M-user downlink system, $\omega(t)$ can represent a vector $(CH_1(t), ..., CH_M(t))$ where $CH_m(t)$ denotes the channel condition of user m at time t.

Denote $\boldsymbol{z}(t) = (\boldsymbol{A}(t), \boldsymbol{e}(t), \omega(t))$. In this paper, we assume that $\boldsymbol{z}(t)$ is i.i.d. and takes values in $\mathcal{Z} = \{\boldsymbol{z}_1, ..., \boldsymbol{z}_K\}$. We then denote $\pi_k = \Pr\{\boldsymbol{z}(t) = \boldsymbol{z}_k\}$. Note that this allows arbitrary dependency among $\boldsymbol{A}(t)$, $\boldsymbol{e}(t)$, and $\omega(t)$.

At every time t, the system operator determines the amount of resource to allocate to serving each queue. We denote this decision by a *matching matrix* $\boldsymbol{b}(t) = (b_{mn}(t), m, n)$, where $b_{mn}(t)$ denotes the type m resource allocated to queue n. When $\boldsymbol{z}(t) = \boldsymbol{z}_k$, $\boldsymbol{b}(t)$ takes values from a finite discrete set $\mathcal{B}_k \subset \mathbb{R}_+^N$.[2]We define $b_{\max} \triangleq \max_{\boldsymbol{b} \in \mathcal{B}_k, k} \|\boldsymbol{b}\|_\infty$ the maximum amount of resource allocated to any queue at any time. Here we define $\|\boldsymbol{x}\|_\infty \triangleq \max_{ij} |x_{ij}|$ for any matrix $\boldsymbol{x}$.[3] It is clear that at any time t, we must have:

$$\sum_n b_{mn}(t) \leq H_m(t), \quad \forall\, m. \tag{1}$$

This is because one cannot spend more resource than what is available. In the following, we call (1) the *no-underflow* constraint. Depending on the system state and resource allocation decision, each task queue gets a service rate $\mu_n(t) \triangleq \mu_n(\boldsymbol{z}(t), \boldsymbol{b}(t))$. We assume $\mu_n(\boldsymbol{z}, \boldsymbol{b}) \in [0, \mu_{\max}]$ for all $\boldsymbol{z}$ and $\boldsymbol{b}$ and that $\{\mu_n(\boldsymbol{z}, \boldsymbol{b})\}_{n=1}^N$ are known to the operator. Also, they satisfy that $\mu_n(\boldsymbol{z}, \boldsymbol{0}) = 0$ for all $\boldsymbol{z}$, and if $\mu_n(\boldsymbol{z}, \boldsymbol{b}) > 0$,

$$\mu_n(\boldsymbol{z}, \boldsymbol{b}) \geq \beta_\mu^l \min_{m: b_{mn} > 0} b_{mn}, \tag{2}$$

for some $\beta_\mu^l > 0$. Moreover, if two vectors $\boldsymbol{b}$ and $\boldsymbol{b}'$ are such that $\boldsymbol{b}'$ is obtained by setting b_{mn} in $\boldsymbol{b}$ to zero, then,

$$\mu_n(\boldsymbol{z}, \boldsymbol{b}) \leq \mu_n(\boldsymbol{z}, \boldsymbol{b}') + \beta_\mu^u b_{mn}, \forall\, n. \tag{3}$$

Note that (2) and (3) are not restrictive. They simply require that nonzero resource is needed for getting a positive service rate, and that a positive rate is upper and lower bounded by linear functions of the resources allocated.

C. Matching Cost and Reward

In every time slot, due to resource expenditure, there is a *matching cost* associated with the resource allocation decision. We model this by denoting $c(t) = c(\boldsymbol{z}(t), \boldsymbol{b}(t))$ the cost for choosing the resource vector $\boldsymbol{b}(t)$. This cost can represent, e.g., cost for purchasing raw materials in inventory control, or payments to workers in a crowdsourcing application. One example is $c(\boldsymbol{z}(t), \boldsymbol{b}(t)) = \sum_{nm} c_m(\boldsymbol{z}(t)) b_{mn}(t)$, where $c_m(\boldsymbol{z}(t))$ denotes the per-unit resource price for type m resource under state $\boldsymbol{z}(t)$. We assume that $c(\boldsymbol{z}, \boldsymbol{b}) \in [0, c_{\max}]$ for all time and it is known to the system operator. Also, if $\boldsymbol{0} \preceq \boldsymbol{b}_1 \preceq \boldsymbol{b}_2$ (entrywise-less), then $c(\boldsymbol{z}, \boldsymbol{b}_1) \leq c(\boldsymbol{z}, \boldsymbol{b}_2)$.

Every time a matching is completed, the operator collects a *matching reward*, e.g., a customer conversion due to an ad, or user satisfaction due to job completion. We model this by denoting the reward collected at time t from type n tasks by $\kappa_n(t)$. We assume that for any time sequence $\{t_j, j = 1, 2, ...\}$ such that $\boldsymbol{z}(t_j) = \boldsymbol{z}$, $\boldsymbol{b}(t_j) = \boldsymbol{b}$ and $\boldsymbol{Q}(t)$, the sequence $\{\kappa_n(t_j) \in [0, r_{\max}], j = 1, 2, ...\}$ consists of i.i.d. random variables, whose mean is determined by the reward function $r_n(\boldsymbol{z}(t), \tilde{\mu}_n(t)) \triangleq \mathbb{E}\big\{\kappa_n(t) \mid \boldsymbol{z}(t), \boldsymbol{b}(t), \boldsymbol{Q}(t)\big\}$. [4] Here $\tilde{\mu}_n(t) = \min[Q_n(t), \mu_n(t)]$ denotes the *actual* amount of tasks completed. We assume that $r_n(\boldsymbol{z}(t), \tilde{\mu}_n(t))$ satisfies:

$$r_n(\boldsymbol{z}(t), \mu) \leq r_n(\boldsymbol{z}(t), \mu'), \text{ if } \mu \leq \mu', \tag{4}$$

[2]This assumption is made to simplify the learning algorithm description (Section VI). Our results can likely be extended to the case when $\{\mathcal{B}_k, k\}$ are general compact sets in $\mathbb{R}_+^N$.

[3]$\boldsymbol{x}$ is a generic symbol that represents a matrix.

[4]The i.i.d. assumption is introduced to facilitate analysis. More general cases can also be considered but will be more involved.

and denote $\boldsymbol{r} = (\boldsymbol{r}_1, ..., \boldsymbol{r}_N)$ the reward matrix, where $\boldsymbol{r}_n = (r_n(\boldsymbol{z}, \mu_n(\boldsymbol{z}, \boldsymbol{b}_z)), \boldsymbol{z} \in \mathcal{Z}, \boldsymbol{b}_z \in \mathcal{B}_{\boldsymbol{z}})$ is a $K \sum |\mathcal{B}_k|$-dimension vector denoting the action-reward correspondence for queue n. Since each $\mathcal{B}_{\boldsymbol{z}}$ is finite, $\boldsymbol{r}$ is also finite.

Different from existing works, e.g., [21], [22], we do not assume any prior knowledge of the function $r_n(\boldsymbol{z}(t), \mu)$.[5] This is common in practice. For example, in crowdsourcing applications, it is often unknown a-prior how qualified a worker is for a certain type of tasks; or in online advertising, one often does not know the conversion probabilities beforehand.

D. Queueing

From the above, we see that the queue vectors $\boldsymbol{Q}(t)$ and $\boldsymbol{H}(t)$ evolve according to:

$$Q_n(t+1) = \max[Q_n(t) - \mu_n(t), 0] + R_n(t), \, \forall \, n, \quad (5)$$

$$H_m(t+1) = H_m(t) - \sum_n b_{mn}(t) + h_m(t), \, \forall \, m, \quad (6)$$

with $\boldsymbol{Q}(t) = \boldsymbol{0}$ and $\boldsymbol{H}(t) = \boldsymbol{0}$. Notice that there is no $\max[\cdot, \cdot]$ operator in (6). This is due to the no-underflow constraint (1). In our paper, we say that a *queue vector process* $\boldsymbol{x}(t) \in \mathbb{R}^d_+$ is stable if it satisfies: [6]

$$x_{\text{av}} \triangleq \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t} \sum_{n=1}^{d} \mathbb{E}\big\{x_n(\tau)\big\} < \infty. \quad (7)$$

E. Utility Optimization

The system's utility is determined by a function of the average matching reward profile. Specifically, define $\overline{r}_n \triangleq \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\big\{r_n(\tau)\big\}$. The system utility is given by:

$$U_{\text{total}}(\overline{\boldsymbol{r}}) \triangleq \sum_n U_n(\overline{r}_n). \quad (8)$$

Here each $U_n(r)$ is an increasing concave function with $U_n(0) = 0$ and $U'(0) < \infty$. We denote $\beta \triangleq \max_{n,r} U'_n(r)$ the maximum first derivative of the utility functions. We also define the following system cost due to resource expenditure:

$$C_{\text{total}} \triangleq \lim_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\big\{c(\tau)\big\}. \quad (9)$$

We say that a matching algorithm Π is feasible if for all time t, it selects $0 \preceq \boldsymbol{R}(t) \preceq \boldsymbol{A}(t)$, $0 \preceq \boldsymbol{h}(t) \preceq \boldsymbol{e}(t)$, and $\boldsymbol{b}(t) \in \mathcal{B}_{\boldsymbol{z}(t)}$, and it ensures constraint (1) for all time. Our objective is to design a feasible policy Π, so as to:

$$\max_{\Pi} : \quad f^{\Pi}_{\text{av}} \triangleq U^{\Pi}_{\text{total}}(\overline{\boldsymbol{r}}) - C^{\Pi}_{\text{total}} \quad (10)$$

$$\text{s.t.} \quad Q_{\text{av}} < \infty, H_{\text{av}} < \infty. \quad (11)$$

Here the superscript Π indicates that the utility and cost are functions of the control policy Π. We denote the optimal solution value as f^*_{av}. Here the queue stability constraints are to ensure that the tasks and resources do not stay in the queue forever. This is important in many cases. For instance, in an energy harvesting network, it is important to ensure timely packet delivery, or in a crowdsourcing system, it is desirable to keep the worker waiting time short.

F. Discussion on the Model

Due to the general matching reward function, our problem is different from a flow utility maximization problem, e.g., [7], which is a special case when $r_n(t) = \tilde{\mu}_n(t)$. By tuning parameters of the model, our model can model all the examples in Section II. For example, by choosing $U_{\text{total}} = \sum_n \alpha_n \overline{r}_n$, the system models the revenue maximziation problem in online advertisement systems. By choosing $\mu_n(t) = 1_{[b_1(t) > b_1^{\min}]} 1_{[b_2(t) > b_2^{\min}]}$, our model can represent a cloud computing system, where a computing task requires two types of resources. Note that in our model we also require the resource queues to be stable in our model for the following two reasons. (i) In certain applications, e.g., crowdsourcing and taxi ride-sharing, the available resource consists of human users who will perform the tasks. In this case, it is important to ensure that workers do not wait indefinitely in the system for tasks. (ii) Imposing the stability constraint helps develop algorithms that only require finite size buffers for holding resources. This is in fact an important feature of our framework. In practice, although it is always beneficial to accumulate more resources, the amount is naturally limited by the finite buffers in the system, and it is often challenging to design algorithms for such systems.

Problem (10) is very challenging. First of all, the no-underflow constraint (1) requires a careful selection of control actions, because actions in a slot can affect action feasibility in later slots. Problems of this kind are often tackled with dynamic programming, whose computational complexity can be extremely high when the action space is large. Secondly, the reward function $r_n(\boldsymbol{z}(t), \tilde{\mu}_n(t))$ is unknown and is dependent on $\boldsymbol{Q}_n(t)$. This makes the problem different from existing utility maximization works, e.g., [7], [8]. Thirdly, due to the more and more stringent user requirements on service quality, it is more desirable to ensure small queueing delay.

Below, we first present an algorithm that assumes full prior knowledge of all system components. This ideal algorithm and its derivation serve as the foundation for the general algorithms. Then, we introduce a general learning model, and present two learning-aided algorithms that explicitly learn and incorporate system information into control. For convenience, we summarize the notations in the paper in Table I.

Notation	Meaning
$\mathcal{Q}_n$ and $Q_n(t)$	Task queue n and its size at time t
$\mathcal{H}_m$ and $H_m(t)$	Resource queue n and its size at time t
$\omega(t)$	System state
$\boldsymbol{z}(t)$	Aggregate system condition
$\boldsymbol{b}(t)$ and $\mathcal{B}_k$	Matching matrix and its feasible set
$\mu_n(\boldsymbol{z}(t), \boldsymbol{b}(t))$	service rate to task queue n
β^u_μ and β^l_μ	Upper and lower slopes of rate functions
$c(\boldsymbol{z}(t), \boldsymbol{b}(t))$	Cost due to resource consumption at time t
$\kappa_n(t)$	Random matching reward for type n tasks
$r_n(\boldsymbol{z}(t), \mu)$	Expected matching reward for type n tasks
$U_n(r_n)$	Utility function for type n tasks
$d_n(t)$	Deficit queue for type n tasks
δ_w	Matrix estimation error

TABLE I
TABLE OF NOTATIONS

IV. MATCHING WITH FULL REWARD INFORMATION

In this section, we will first present an ideal algorithm, which assumes full knowledge of the reward functions

[5]This is different from the $\boldsymbol{\mu}$ functions, which measure how much resources are spent and can typically be observed by the system controller.

[6]In this paper, we assume that all limits exist with probability 1. Our results can be extended to more general cases with $\liminf$ or $\limsup$ arguments.

$r_n(\boldsymbol{z}(t),\mu)$ and will serve as a basic building block. Even in this case, we will see that the problem is highly nontrivial due to the existence of the no-underflow constraint (1) and the dependency of $r_n(t)$ on $\boldsymbol{Q}(t)$.

To start, we first introduce an auxiliary variable $\gamma_n(t) \in [0, r_{\max}]$ and create for each n a *deficit queue* $d_n(t)$ that evolves as follows:

$$d_n(t+1) = \max[d_n(t) - \kappa_n(t), 0] + \gamma_n(t), \qquad (12)$$

with $\boldsymbol{d}(0) = \boldsymbol{0}$. Note that the input into $d_n(t)$ is $\kappa_n(t)$ instead of $r_n(t)$. The deficit queue $d_n(t)$ measures how much the actual reward profile is currently lagging behind the target value (due to randomness). Hence, stabilizing the deficit queues implies that the average reward collected from type n tasks is no smaller than the target value.

We first present our algorithm assuming full knowledge of the expected reward vector $\boldsymbol{r}(t)$.

Reward optimAl Matching (RAM): At every time t, observe $\boldsymbol{z}(t)$ and $\boldsymbol{y}(t)$. Do:

1) **Quota:** For each n, choose $\gamma_n(t)$ by solving:

$$\max: \ VU_n(\gamma_n(t)) - d_n(t)\gamma_n(t), \ \text{s.t. } 0 \le \gamma_n(t) \le r_{\max}. \ (13)$$

2) **Admission:** For each n, if $Q_n(t) < \theta_1$, let $R_n(t) = A_n(t)$; otherwise $R_n(t) = 0$. Similarly, for each m, if $H_m(t) < \theta_2$, let $h_m(t) = e_m(t)$; otherwise $h_m(t) = 0$.
3) **Resource:** Choose the resource allocation vector $\boldsymbol{b}(t)$ by solving:

$$\begin{aligned}\min: \ \Psi_r(\boldsymbol{b}) \triangleq Vc(\boldsymbol{z}(t),\boldsymbol{b}) &- \sum_m (H_m(t)-\theta_2)\sum_n b_{mn} \quad (14)\\ &- \sum_n (Q_n(t)-\theta_1)\mu_n(\boldsymbol{z}(t),\boldsymbol{b})\\ &- \sum_n d_n(t) r_n(\boldsymbol{z}(t),\mu_n(\boldsymbol{z}(t),\boldsymbol{b}))\\ \text{s.t. } \ \boldsymbol{b} \in \mathcal{B}_{\boldsymbol{z}(t)}&, \text{Constraint (1)} \qquad (15)\end{aligned}$$

4) **Queueing:** The queues $\boldsymbol{Q}(t)$, $\boldsymbol{H}(t)$, and $\boldsymbol{d}(t)$ evolve according to (5), (6), and (12), respectively. $\Diamond$

RAM takes as input the state $\boldsymbol{z}(t)$ and the queue vector $\boldsymbol{y}(t)$, and outputs corresponding actions $\boldsymbol{\gamma}(t)$, $\boldsymbol{R}(t)$, and $\boldsymbol{b}(t)$. In the algorithm, V is a parameter introduced for controlling the tradeoff between system utility and service delay (explained later). Note that in (14) we have used $r_n(\boldsymbol{z}(t),\mu_n(t))$ instead of $r_n(\boldsymbol{z}(t),\tilde{\mu}_n(t))$. We will see in our later analysis that our algorithm automatically guarantees $\tilde{\mu}_n(t) = \mu_n(t)$. This is very useful, for otherwise the algorithm performance will be very hard to analyze. We also emphasize here that the introduction of θ_1 and θ_2 are important. It can be seen in the admission step that if $\theta_1 = \theta_2 = 0$, no task or resource will be admitted at the first place and the algorithm will not even proceed! From the algorithm description, we see how the $r_n(\boldsymbol{z}(t),\mu_n(t))$ functions come into the picture. The explicit form of (14) also enables an exact analysis about how estimation errors in $r_n(\boldsymbol{z}(t),\mu_n(t))$ affect performance.

We now provide the mathematical derivation behind RAM. To do so, we denote

$$f_\gamma(t) \triangleq \sum_n U_n(\gamma_n(t)) - c(t) \qquad (16)$$

the *target* instantaneous system utility minus cost and denote $\boldsymbol{y}(t) \triangleq (\boldsymbol{Q}(t),\boldsymbol{H}(t),\boldsymbol{d}(t))$. We then define a Lyapunov function as follows:

$$L(t) = \frac{1}{2}\|\boldsymbol{Q}(t)-\boldsymbol{\theta}_1\|^2 + \frac{1}{2}\|\boldsymbol{H}(t)-\boldsymbol{\theta}_2\|^2 + \frac{1}{2}\|\boldsymbol{d}(t)\|^2, \ (17)$$

where $\|\cdot\|$ is the euclidean norm and $\boldsymbol{\theta}_1 = \theta_1\cdot\boldsymbol{1}^N$ and $\boldsymbol{\theta}_2 = \theta_2\cdot\boldsymbol{1}^M$ with $\boldsymbol{1}^k \in \mathbb{R}^k$ being the vector with all components being 1, and θ_1 and θ_2 are constants that will be specified later. We define the one-slot utility-based conditional Lyapunov drift $\Delta_V(t) \triangleq \mathbb{E}\big\{L(t+1)-L(t)-Vf_\gamma(t) \mid \boldsymbol{y}(t)\big\}$. Using queueing dynamics (5), (6), and (12), we obtain the following lemma, in which $V \ge 1$ is the tunable parameter as in the RAM algorithm.

Lemma 1. *Under any feasible policy, we have:*

$$\begin{aligned}\Delta_V(t) \le G - V\sum_n \mathbb{E}\big\{U_n(\gamma_n(t)) - d_n(t)\gamma_n(t) \mid \boldsymbol{y}(t)\big\} \quad (18)\\ +\sum_n (Q_n(t)-\theta_1)\mathbb{E}\big\{R_n(t) \mid \boldsymbol{y}(t)\big\}\\ +\sum_m (H_m(t)-\theta_2)\mathbb{E}\big\{h_m(t) \mid \boldsymbol{y}(t)\big\}\\ +\mathbb{E}\big\{Vc(t) - \sum_m (H_m(t)-\theta_2)\sum_n b_{mn}(t)\\ -\sum_n (Q_n(t)-\theta_1)\mu_n(t) - \sum_n d_n(t) r_n(t) \mid \boldsymbol{y}(t)\big\}.\end{aligned}$$

Here $G \triangleq N(A^2_{\max}+\mu^2_{\max}+2r^2_{\max}) + Mh^2_{\max} + MN^2b^2_{\max}$ does not depend on V, and the expectations are taken over the randomness in the system as well as in the policy. $\Diamond$

Proof. See Appendix A. □

V. Learning Module

From the previous section, we see that if full reward information is available beforehand, RAM can be used to achieve optimal utility. In the following, we turn to consider the case when such information is not given and must be learned. To investigate the impact of learning on algorithm design and performance, we first define the notion of learning capability. Specifically, for any general matrix $\boldsymbol{W}$ and a learning algorithm Γ that outputs an estimation $\hat{\boldsymbol{W}}$, we denote its maximum estimation error by:

$$\delta_w \triangleq \|\hat{\boldsymbol{W}} - \boldsymbol{W}\|_\infty. \qquad (19)$$

Recall that $\|\boldsymbol{x}\|_\infty \triangleq \max_{ij}|x_{ij}|$. Then, the formal definition of a *learning module* is as follows.

Definition 1. *An algorithm Γ is called a $(T_\delta, P_\delta, \delta)$-learning module, if (i) it completes learning in T_δ time, (ii) it guarantees $Pr\big\{\delta_w < \delta\big\} \ge P_\delta$, and (iii) for any $T \ge 0$, P_δ does not decrease if the algorithm is run for $T_\delta + T$ time.* $\Diamond$

Here δ specifies the estimation accuracy guarantee, and T_δ can be both random or deterministic depending on the termination rules. This definition is general and captures key features of learning algorithms. With this definition, having perfect knowledge at the beginning can be viewed as having an $(0,1,0)$-learning module. The introduction of a learning module is to enable an explicit separation between the learning component and the control component. Doing so provides new guideline for developing a modular design procedure for learning-aided control algorithms. We will also show how parameters of the learning module can be incorporated into system control and how they affect system performance.

As concrete examples, we now describe two sampling-based learning modules for estimating $\boldsymbol{r}$ and $\boldsymbol{\pi}$. We first describe a threshold-based sampling module for estimating $\boldsymbol{r}$. In the module, we use $s(\boldsymbol{z}, \boldsymbol{b}, t)$ to denote the number of times the pair $(\boldsymbol{z}, \boldsymbol{b})$ is sampled, i.e., adopt $\boldsymbol{b}$ when $\boldsymbol{z}(t) = \boldsymbol{z}$, up to time t. We also denote $s_{\min}(t) = \min_{(\boldsymbol{z},\boldsymbol{b})} s(\boldsymbol{z}, \boldsymbol{b}, t)$.

Threshold-Based Sampling $\texttt{TBS}(s_{th})$: Every time t, sample the resource allocation vector $\boldsymbol{b}^* \in \arg\min_{\boldsymbol{b}} s(\boldsymbol{z}(t), \boldsymbol{b}, t)$ until $s_{\min}(t) \geq s_{th}$. If terminate at T_{tbs}, output $\hat{r}_n(\boldsymbol{z}, \boldsymbol{b}) = \sum_{t=1}^{T_{tbs}} 1_{[\boldsymbol{z}(t)=\boldsymbol{z},\boldsymbol{b}(t)=\boldsymbol{b}]}\kappa_n(t)/s(\boldsymbol{z}, \boldsymbol{b}, T_{tbs})$. $\Diamond$

Here $1_{[x]}$ is the indication function of x. The learning module $\texttt{TBS}(s_{th})$ is very intuitive. It tries to balance the sampling frequencies of all $(\boldsymbol{z}, \boldsymbol{b})$ until every pair is sampled at least s_{th} times. In this case, the learning algorithm running time T_{tbs} is random. In the following, we look at a deterministic time sampler for estimating $\boldsymbol{\pi}$. This module sets a sampling time threshold T_{th}.

Time-Limited Sampling $\texttt{TLS}(T_{th})$: Observe $\boldsymbol{z}(t)$ for T_{th} slots. Output $\hat{\pi}_k = \sum_{t=1}^{T_{th}} 1_{[\boldsymbol{z}(t)=\boldsymbol{z}_k]}/T_{th}$ for all k. $\Diamond$

The following lemma shows the performance of the two modules.

Lemma 2. *The two learning modules satisfy:*

(a) $\texttt{TBS}(s_{th})$*:* $\mathbb{E}\{T_{tbs}\} = \Theta(s_{th})$ *and we have with probability* $1 - O(e^{-\frac{s_{th}\log(s_{th})^2}{2(s_{th}r_{\max}^2 + r_{\max}\sqrt{s_{th}}/3)}})$ *that* $\delta_r = \frac{\log(s_{th})}{\sqrt{s_{th}}}$.

(b) $\texttt{TLS}(T_{th})$*: With probability* $1 - O(e^{-\frac{T^{th}\log(T_{th})^2}{2(T_{th}+\sqrt{T_{th}}/3)}})$, $\delta_\pi = \frac{\log(T_{th})}{\sqrt{T_{th}}}$. $\Diamond$

Proof. See Appendix B. □

By choosing $s_{th} = T_{th} = V^c$, we can guarantee $\delta_r = \delta_\pi = c\log(V)V^{-c/2}$ with P_{δ_r} and P_{δ_π} being $1 - O(V^{-\log(V)})$.

VI. Optimal Matching With Learning

We now consider the case when one does not have full reward information and present two algorithms that integrate general learning methods for estimating $\boldsymbol{r}$ and $\boldsymbol{\pi}$.

A. With Reward Information Learning

We first present an optimal matching algorithm for general systems that do not possess perfect knowledge of $\boldsymbol{r}$ and need to rely on some learning algorithms for estimation. In the algorithm, we use $\beta_{\hat{r}}$ to denote the maximum "derivative" of the estimated $\hat{\boldsymbol{r}}$ with respect to any b_{mn}. Specifically, we assume that if $\boldsymbol{b}$ and $\boldsymbol{b}'$ are such that $\boldsymbol{b}'$ is obtained by setting one b_{mn} in $\boldsymbol{b}$ to zero. Then,

$$\hat{r}_n(\boldsymbol{z}, \mu_n(\boldsymbol{z}, \boldsymbol{b})) \quad \leq \quad \hat{r}_n(\boldsymbol{z}, \mu_n(\boldsymbol{z}, \boldsymbol{b}')) + \beta_{\hat{r}} b_{mn}, \forall n. \quad (20)$$

Since both $\mathcal{Z}$ and $\{\mathcal{B}_{\boldsymbol{z}}\}$ are finite, we see that $\beta_{\hat{r}}$ exists and is $\Theta(1)$ (possibly depends on $\hat{\boldsymbol{r}}$).

Learning-aided Reward OptimAl Matching ($\texttt{LRAM}$):

1) (**Learning**) Apply any $(T_{\delta_r}, P_{\delta_r}, \delta_r)$-learning module Γ_r. Terminate at $t = T_{\delta_r}$ and output $\hat{\boldsymbol{r}}$.
2) (**Matching**) Set $\boldsymbol{Q}(T_{\delta_r} + 1) = 0$, $\boldsymbol{H}(T_{\delta_r} + 1) = 0$, and $\boldsymbol{d}(T_{\delta_r} + 1) = 0$. Choose θ_1 and θ_2 according to:

$$\theta_1 = (h_{\max} + (V\beta + r_{\max})\beta_{\hat{r}})/\beta_\mu^l + \mu_{\max} \quad (21)$$
$$\theta_2 = (V\beta + r_{\max})\beta_{\hat{r}} + r_{\max}\beta_\mu^u + Nb_{\max}. \quad (22)$$

Run $\texttt{RAM}$ with $\hat{\boldsymbol{r}}$, i.e., replace $r_n(\boldsymbol{z}(t), \mu_n(\boldsymbol{z}(t), \boldsymbol{b}))$ in $\texttt{RAM}$ by $\hat{r}_n(\boldsymbol{z}(t), \mu_n(\boldsymbol{z}(t), \boldsymbol{b}))$. $\Diamond$

In $\texttt{LRAM}$, we explicitly separate the algorithm into two disjoint phases. This is chosen to facilitate presentation and analysis. Doing so does not change the order of the overall algorithm convergence time and performance. We can also transform $\texttt{LRAM}$ and $\texttt{DRAM}$ below into continuous-learning versions, e.g., [17], and update estimations from time to time, e.g., using sliding-window estimation or frame-based estimation. Lastly, it is worth noting that θ_1 and θ_2 can be computed beforehand easily. This is a feature useful for implementation.

B. With System State Information Learning

In the previous section, we describe how the estimated reward information $\hat{\boldsymbol{r}}$ can be naturally integrated into a matching algorithm. Here we consider the case when a learning module is also applied to learning the statistics of the system state $\boldsymbol{z}(t)$. Our result here generalizes the dual-learning approach proposed in [17] to handle underflow and to allow general learning methods.

To start, we define the following optimization problem:

$$\max: \quad \Phi \triangleq V[\sum_n U_n(\gamma_n) - Cost] \quad (23)$$
$$\text{s.t.} \quad \gamma_n \leq r_n \triangleq \sum_k \pi_k r_n(\boldsymbol{z}_k, \mu_n(\boldsymbol{z}_k, \boldsymbol{b}^k)) \quad (24)$$
$$Cost \triangleq \sum_k \pi_k c(\boldsymbol{z}_k, \boldsymbol{b}^k) \quad (25)$$
$$\sum_k \pi_k R_n^k = \sum_k \pi_k \mu_n(\boldsymbol{z}_k, \boldsymbol{b}^k), \forall n \quad (26)$$
$$\sum_k \pi_k \sum_n b_{mn}^k = \sum_k \pi_k h_m^k, \forall m \quad (27)$$
$$0 \leq \gamma_n \leq r_{\max}, \boldsymbol{b}^k \in \mathcal{B}_k \quad (28)$$
$$0 \preceq \boldsymbol{R}^k \preceq \boldsymbol{A}^k, 0 \preceq \boldsymbol{h}^k \preceq \boldsymbol{e}^k. \quad (29)$$

Problem (23) can intuitively be viewed as a way to solve our matching problem.[7] $\boldsymbol{\gamma}$ is introduced to decouple the randomness and the nonlinear objective function. The equalities in (26) and (27) are due to fact that only tasks that are actually served generate reward and the no-underflow constraint (1). However, a scheme obtained by solving (23) may not be implementable due to (i) it ignores the no-underflow constraint, and (ii) it assumes that all resources allocated are fully utilized, i.e., using $\mu_n(z_k, \boldsymbol{b}^k)$ in (24). We will also see later that, it may require a much larger learning time for such a scheme to have the right statistics for achieving a performance comparable to our schemes.

We now obtain the dual problem for (23) as follows.

$$\min: \quad g(\boldsymbol{\alpha}^d, \boldsymbol{\alpha}^q, \boldsymbol{\alpha}^h) \text{ s.t. } \boldsymbol{\alpha}^d \succeq \boldsymbol{0}, \boldsymbol{\alpha}^q \in \mathbb{R}^N, \boldsymbol{\alpha}^h \in \mathbb{R}^M, \quad (30)$$

where $g(\boldsymbol{\alpha}^d, \boldsymbol{\alpha}^q, \boldsymbol{\alpha}^h) = \sum_k \pi_k g_k(\boldsymbol{\alpha}^d, \boldsymbol{\alpha}^q, \boldsymbol{\alpha}^h)$ is the dual function, and $g_k(\boldsymbol{\alpha}^d, \boldsymbol{\alpha}^q, \boldsymbol{\alpha}^h)$ is defined as:

$$g_k(\boldsymbol{\alpha}^d, \boldsymbol{\alpha}^q, \boldsymbol{\alpha}^h) = \sup_{\boldsymbol{R},\boldsymbol{\gamma},\boldsymbol{b},\boldsymbol{h}} \Big\{ V[\sum_n U_n(\gamma_n) - c(\boldsymbol{z}_k, \boldsymbol{b}^k)] \quad (31)$$
$$- \sum_n \alpha_n^d [r_n(\boldsymbol{z}_k, \mu_n(\boldsymbol{z}_k, \boldsymbol{b})) - \gamma_n]$$

[7]Technically speaking, one has to solve a convexified version of the problem in order to find an optimal control policy for our problem.

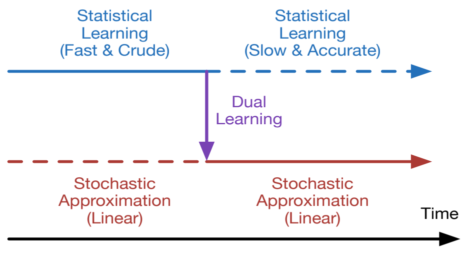

Statistical Learning (Fast & Crude)
Statistical Learning (Slow & Accurate)
Dual Learning
Stochastic Approximation (Linear)
Stochastic Approximation (Linear)
Time

Fig. 2. DRAM combines the fast regime of statistical learning and the smoothness of queue-based control (more generally, stochastic approximation).

$$-\sum_n \alpha_n^q[R_n-\mu_n(\boldsymbol{z}_k,\boldsymbol{b})]-\sum_m \alpha_m^h[\sum_n b_{mn}-h_n]\Big\}.$$

Note that $g_k(\boldsymbol{\alpha}^d,\boldsymbol{\alpha}^q,\boldsymbol{\alpha}^h)$ is the dual function for state $\boldsymbol{z}_k$.

With (30), below we present our algorithm, which also performs system state information learning. The main idea of the algorithm is to first utilize learning to obtain an empirical distribution (in addition to learning $\boldsymbol{r}$), which is usually crude but fast at the beginning. Then, it transforms to queue-based control (or more generally, stochastic approximation), by obtaining an empirical optimal multiplier via dual learning. It then starts from the empirical multiplier and rely on queue-based control to learn the true optimal operation point. The procedure is shown in Fig. 2. This combination avoids the slow convergence regime of statistical learning and the slow start of stochastic approximation, and algorithms so developed can achieve much faster convergence and superior delay.

The formal algorithm is as follows.

Dual learning-aided Reward optimAl Matching (DRAM):

1) (**Learning**) Apply any $(T_{\delta_r},P_{\delta_r},\delta_r)$-learning module Γ_r for $\boldsymbol{r}$ and any $(T_{\delta_\pi},P_{\delta_\pi},\delta_\pi)$-learning module Γ_z for $\boldsymbol{z}$. Terminate at $T_L=\max(T_{\delta_r},T_{\delta_\pi})$ and output $\hat{\boldsymbol{r}}$ and $\hat{\boldsymbol{\pi}}$. Choose θ_1 and θ_2 according to:
$$\theta_1 = (h_{\max}+(V\beta+r_{\max})\beta_{\hat{r}})/\beta_\mu^l+\mu_{\max} \quad (32)$$
$$\theta_2 = (V\beta+r_{\max})\beta_{\hat{r}}+r_{\max}\beta_\mu^u+Nb_{\max}. \quad (33)$$
2) (**Dual learning**) Solve the *empirical dual problem*:
$$\min:\ \sum_k \hat{\pi}_k\hat{g}_k(\boldsymbol{\alpha}^d,\boldsymbol{\alpha}^q-\boldsymbol{\theta}_1,\boldsymbol{\alpha}^h-\boldsymbol{\theta}_2) \quad (34)$$
$$\text{s.t.}\quad \boldsymbol{\alpha}^d\succeq\boldsymbol{0},\boldsymbol{\alpha}^q\in\mathbb{R}^N,\boldsymbol{\alpha}^h\in\mathbb{R}^M.$$
Here $\hat{g}_k(\boldsymbol{\alpha}^d,\boldsymbol{\alpha}^q-\boldsymbol{\theta}_1,\boldsymbol{\alpha}^h-\boldsymbol{\theta}_2)$ is defined in (31) with $\hat{\boldsymbol{r}}$. Denote the optimal solution as $\hat{\boldsymbol{\alpha}}^*=(\hat{\boldsymbol{\alpha}}^{d*},\hat{\boldsymbol{\alpha}}^{q*},\hat{\boldsymbol{\alpha}}^{h*})$.
3) (**Matching**) Set $\boldsymbol{Q}(T_L+1)=0$, $\boldsymbol{H}(T_L+1)=0$, and $\boldsymbol{d}(T_L+1)=0$. For all $t\geq T_L+1$, define:
$$\hat{\boldsymbol{Q}}(t) = \boldsymbol{Q}(t)+\hat{\boldsymbol{\alpha}}^{q*}-\boldsymbol{\zeta}_N \quad (35)$$
$$\hat{\boldsymbol{H}}(t) = \boldsymbol{H}(t)+\hat{\boldsymbol{\alpha}}^{h*}-\boldsymbol{\zeta}_M \quad (36)$$
$$\hat{\boldsymbol{d}}(t) = \boldsymbol{d}(t)+\hat{\boldsymbol{\alpha}}^{d*}-\boldsymbol{\zeta}_N, \quad (37)$$
where $\boldsymbol{\zeta}_k\in\mathbb{R}^k$ has all entries being $\zeta\triangleq 2\max(\delta_\pi V\log(V)^2,\log(V)^2)$. Run RAM with $\hat{\boldsymbol{r}}$, $\hat{\boldsymbol{Q}}(t)$, $\hat{\boldsymbol{H}}(t)$, and $\hat{\boldsymbol{d}}(t)$, i.e., replace $r_n(\boldsymbol{z}(t),\mu_n(\boldsymbol{z}(t),\boldsymbol{b}))$, $\boldsymbol{Q}(t)$, $\boldsymbol{H}(t)$, and $\boldsymbol{d}(t)$ in RAM by $\hat{r}_n(\boldsymbol{z}(t),\mu_n(\boldsymbol{z}(t),\boldsymbol{b}))$, $\hat{\boldsymbol{Q}}(t)$, $\hat{\boldsymbol{H}}(t)$, and $\hat{\boldsymbol{d}}(t)$, respectively. If $\boldsymbol{b}(t)$ from (14) violates (1) for some m, change $\{b_{mn}(t)\}_{n\in\mathcal{N}}$ to $\{\tilde{b}_{mn}(t)\}_{n\in\mathcal{N}}$ with $\sum_n\tilde{b}_{mn}(t)=h_m(t)$ and drop $\mu_n(\boldsymbol{z}(t),\boldsymbol{b}(t))$ tasks from each n that has $b_{mn}(t)>0$.[8]$\Diamond$

[8]In actual implementation, one can still serve the tasks with the actual allocated resource.

Note here that the actual queues $\boldsymbol{Q}(t)$, $\boldsymbol{H}(t)$, and $\boldsymbol{d}(t)$ still evolve according to (5), (6), and (12), respectively, and DRAM defines $\hat{\boldsymbol{Q}}(t)$, $\hat{\boldsymbol{H}}(t)$, and $\hat{\boldsymbol{d}}(t)$ on top of their values. The dropping step is introduced to ensure zero underflow, by giving up the service rates and reward at that particular slot. We will see in later analysis and simulation that such an event almost never happens and hence does not affect performance.

We note that both LRAM and DRAM are two-stage algorithms, which commonly appear in the multi-armed bandit literature, e.g., [15], [16]. However, our formulation and solutions are different in that (i) queue underflow affects reward, and (ii) queueing delay is explicitly considered and analyzed.

VII. Performance Analysis

In this section, we present the performance results of LRAM and DRAM. We start with some definitions and assumptions. For notation simplicity, we denote $\boldsymbol{\alpha}=(\boldsymbol{\alpha}^d,\boldsymbol{\alpha}^q,\boldsymbol{\alpha}^h)$ and write $\boldsymbol{\alpha}-\boldsymbol{\theta}=(\boldsymbol{\alpha}^d,\boldsymbol{\alpha}^q-\boldsymbol{\theta}_1,\boldsymbol{\alpha}^h-\boldsymbol{\theta}_2)$. Also, to indicate the different distributions and reward functions used, we use $\tilde{g}(\boldsymbol{\alpha})$ to denote the dual function when $\boldsymbol{r}$ is replaced by $\hat{\boldsymbol{r}}$ in (31) and the distribution is given by $\boldsymbol{\pi}$. Then, we use $g^{\hat{\pi}}(\boldsymbol{\alpha})$ and $\hat{g}^{\hat{\pi}}(\boldsymbol{\alpha})$ to denote the dual function with distribution $\hat{\boldsymbol{\pi}}$ and $\boldsymbol{r}$, and with $\hat{\boldsymbol{\pi}}$ and $\hat{\boldsymbol{r}}$, respectively.

A. Preliminaries

We define the following polyhedral system structure:

Definition 2. *A system is polyhedral with parameter $\rho>0$ if the dual function $g(\boldsymbol{\alpha})$ satisfies:*
$$g(\boldsymbol{\alpha}^*)\leq g(\boldsymbol{\alpha})-\rho\|\boldsymbol{\alpha}^*-\boldsymbol{\alpha}\|. \quad (38)$$
Here $\boldsymbol{\alpha}^*$ is an optimal solution of (30). $\Diamond$

The polyhedral structure typically appears in practical systems, especially when system action sets are discrete (see [23] for more discussions). Note that (38) holds for all V if it holds under any V, in particular $V=1$.

In our analysis, we make the following assumptions.

Assumption 1. *There exist constants $\epsilon_r,\epsilon_z=\Theta(1)>0$ such that for any valid state distribution $\hat{\boldsymbol{\pi}}$ and reward statistics $\hat{\boldsymbol{r}}$ with $\|\hat{\boldsymbol{\pi}}-\boldsymbol{\pi}\|\leq\epsilon_z$ and $||\boldsymbol{r}-\hat{\boldsymbol{r}}||\leq\epsilon_r$, there exist a set of actions $\{\boldsymbol{\gamma}^k\}_{k=1,...,|\mathcal{Z}|}$, $\{\boldsymbol{R}_i^k\}_{k=1,...,|\mathcal{Z}|}^{i=1,2,...,\infty}$, $\{\boldsymbol{b}_i^k\}_{k=1,...,|\mathcal{Z}|}^{i=1,2,...,\infty}$, and $\{\boldsymbol{h}_i^k\}_{k=1,...,|\mathcal{Z}|}^{i=1,2,...,\infty}$, and variables $\lambda_i^k\geq 0$ with $\sum_i\lambda_i^k=1$ for all k (possibly depending on $\hat{\boldsymbol{\pi}}$ and $\hat{\boldsymbol{r}}$), such that:*
$$\gamma_n-\sum_k\hat{\pi}_k\sum_i\lambda_i^k\hat{r}_n(\boldsymbol{z}_k,\mu_n(z_k,\boldsymbol{b}_i^k))\leq-\eta_0, \quad (39)$$
where $\eta_0=\Theta(1)>0$ is independent of $\hat{\boldsymbol{\pi}}$ and $\hat{\boldsymbol{r}}$, and that
$$\sum_k\hat{\pi}_k\sum_i\lambda_i^kR_{in}^k=\sum_k\hat{\pi}_k\sum_i\lambda_i^k\mu_n(\boldsymbol{z}_k,\boldsymbol{b}_i^k) \quad (40)$$
$$\sum_k\hat{\pi}_k\sum_i\lambda_i^k\sum_n b_{imn}^k=\sum_k\hat{\pi}_k\sum_i\lambda_i^kh_{im}^k \quad (41)$$
where $0<\sum_k\hat{\pi}_k\sum_i\lambda_i^kR_{in}^k<\mathbb{E}_{\hat{\pi}}\{A_n(t)\}$ as well as $0<\sum_k\hat{\pi}_k\sum_i\lambda_i^kh_{im}^k<\mathbb{E}_{\hat{\pi}}\{e_m(t)\}$. $\Diamond$

Assumption 2. *For any $\hat{\boldsymbol{\pi}}$ and $\hat{\boldsymbol{r}}$ with $\|\hat{\boldsymbol{\pi}}-\boldsymbol{\pi}\|\leq\epsilon_z$ and $||\hat{\boldsymbol{r}}-\boldsymbol{r}||\leq\epsilon_r$, if $g(\boldsymbol{\alpha})$ is polyhedral with parameter ρ, $\hat{g}^{\hat{\pi}}(\boldsymbol{\alpha})$ is also polyhedral with parameter ρ.* $\Diamond$

Assumption 3. *For any $\hat{\boldsymbol{\pi}}$ and $\hat{\boldsymbol{r}}$ with $\|\hat{\boldsymbol{\pi}}-\boldsymbol{\pi}\| \leq \epsilon_z$ and $\|\hat{\boldsymbol{r}}-\boldsymbol{r}\| \leq \epsilon_r$, $\hat{g}^{\hat{\pi}}(\boldsymbol{\alpha})$ has a unique optimal solution over $\mathbb{R}^{M+2N}$.* $\Diamond$

In the network optimization literature, e.g., [12], [24], Assumption 1 is commonly assumed with $\epsilon_r = \epsilon_z = 0$. By allowing $\epsilon_r, \epsilon_z > 0$, we assume that systems with similar statistics have similar stability regions. Assumption 2 assumes that systems with similar statistics share a similar dual structure. This is not restrictive. In fact, when action sets are discrete, it is often the case that $g_k(\boldsymbol{\alpha})$ is polyhedral, which usually leads to a polyhedral structure of $\hat{g}^{\hat{\pi}}(\boldsymbol{\alpha})$. The uniqueness assumption is also often guaranteed by the utility maximization structure, e.g., [25].

Below, we adopt the following big-O notations taken from Chapter 3 in [26]. Specifically, we say that $f(x) = O(g(x))$ if there exists a constant $c_1, x_1 < \infty$ such that $0 \leq f(x) \leq c_1 g(x)$ for all $x \geq x_1$, and that $f(x) = \Theta(g(x))$ if $f(x) = O(g(x))$ and $g(x) = O(f(x))$. In this work, the notations are defined with respect to the parameter V.

B. Queue and Utility Performance

To state our results, we define the following events about the correctness of the estimations of the reward and distribution functions, i.e.,

$$\mathcal{E}_r = \{\|\hat{\boldsymbol{r}}-\boldsymbol{r}\| \leq \delta_r\},\ \ \mathcal{E}_\pi = \{\|\hat{\boldsymbol{r}}-\boldsymbol{r}\| \leq \delta_r\}. \tag{42}$$

We first summarize the performance of LRAM.

Theorem 1. *Suppose $T_{\delta_r} < \infty$ with probability 1. Under* LRAM*, we have for all $t \geq T_{\delta_r}+1$ that:*

$$d_n(t) \ \leq\ d_{\max} \triangleq V\beta + r_{\max},\ \forall n \tag{43}$$

$$Q_n(t) \ \leq\ Q_{\max} \triangleq \theta_1 + r_{\max},\ \forall n \tag{44}$$

$$H_m(t) \ \leq\ H_{\max} \triangleq \theta_2 + h_{\max},\ \forall m. \tag{45}$$

Moreover, with probability P_{δ_r}, $\mathcal{E}_r$ occurs, and

$$f^{\texttt{LRAM}}_{av,\mathcal{E}_r} \geq f^*_{av} - \frac{G + r_{\max}\delta_r}{V} - 2N\beta\delta_r. \tag{46}$$

Here $f^{\texttt{LRAM}}_{av,\mathcal{E}_r}$ denotes the resulting system performance under LRAM*, as defined in (10) conditioning on $\mathcal{E}_r$, i.e., $\overline{r}_n = \lim_{t\to\infty}\frac{1}{t}\sum_{\tau=T_{\delta_r}}^{t-1}\mathbb{E}\{r_n(\tau)\,|\,\mathcal{E}_r\}$ and $C_{total} = \lim_{t\to\infty}\frac{1}{t}\sum_{\tau=T_{\delta_r}}^{t-1}\mathbb{E}\{c(\tau)\,|\,\mathcal{E}_r\}$.* $\Diamond$

Proof. See Appendix C. $\square$

Theorem 1 shows that LRAM guarantees deterministic queueing bounds of size $O(V)$, and the utility loss (compared to the optimal value) is $O(1/V+\delta_r)$, which can be viewed as the performance loss due to inaccurate reward information. We remark here that the deterministic bounds are important for both algorithm implementation and performance guarantees. This is because errors in reward estimation will be amplified by the queue sizes when used in decision making, i.e., (14). Note that if we have $\delta_r = 0$ with $T_{\delta_r} = 0$ and $P_{\delta_r} = 1$, then Theorem 1 recovers the known $[O(1/V), O(V)]$ utility-delay tradeoff for stochastic network problems [12].

We now present the performance results for DRAM.

Theorem 2. *Suppose $\max(T_{\delta_r}, T_{\delta_\pi}) < \infty$ with probability 1. Suppose $g(\boldsymbol{\alpha})$ is polyhedral with $\rho = \Theta(1) > 0$, and that $\delta_\pi \leq \epsilon_z$ and $\delta_r \leq \epsilon_r$, and $\boldsymbol{\alpha}^* + \boldsymbol{\theta} \succ \mathbf{0}$. Then, with a sufficiently large V, we have with probability $P_{\delta_\pi}P_{\delta_r}$ that both learning modules succeed, i.e., $\mathcal{E}_r \cap \mathcal{E}_\pi$ occurs, and that under* DRAM*,*

$$f^{\texttt{DRAM}}_{av,\mathcal{E}_r\cap\mathcal{E}_\pi} \geq f^*_{av} - \frac{G}{V} - O(1/V + \delta_r). \tag{47}$$

Here $f^{\texttt{DRAM}}_{av,\mathcal{E}_r\cap\mathcal{E}_\pi}$ is defined with $\overline{r}_n$ and C_{total} being the average expected value conditioning on $\mathcal{E}_r \cap \mathcal{E}_\pi$. Also, the fraction of time dropping happens is $O(V^{-\log(V)})$. Moreover, for all queues, there exist $\Theta(1)$ constants D, K, a such that:

$$\lim_{t\to\infty} Pr\{d_n(t) \geq \frac{3}{2}\zeta + D + \nu\} \ \leq\ ae^{-K\nu} \tag{48}$$

$$\lim_{t\to\infty} Pr\{Q_n(t) \geq \frac{3}{2}\zeta + D + \nu\} \ \leq\ ae^{-K\nu} \tag{49}$$

$$\lim_{t\to\infty} Pr\{H_m(t) \geq \frac{3}{2}\zeta + D + \nu\} \ \leq\ ae^{-K\nu}. \tag{50}$$

Thus, all queues are stable. $\Diamond$

Proof. See Appendix D. $\square$

Theorem 2 shows that the utility performance of DRAM is similarly within $O(1/V+\delta_r)$ of the optimal. However, instead of showing deterministic bounds as in Theorem 1, it shows that the queues will rarely go beyond $\zeta = O(\delta_\pi V\log(V)^2 + \log(V)^2)$. This provides additional insight into how learning accuracy affects delay performance. Similar to the LRAM case, if we also have $\delta_\pi = 0$ with $T_{\delta_\pi} = 0$ and $P_{\delta_\pi} = 1$, then DRAM provides a new way for achieving the near-optimal $[O(1/V), O(\log(V)^2)]$ utility-delay tradeoff.

In both LRAM and DRAM, it is possible to continuously update the reward function estimations during control steps. However, this does not automatically guarantee that we can always eliminate the effect of δ_r. This is so because the initial estimation $\hat{\boldsymbol{r}}$ may affect what options will be continuously updated later. On the other hand, the same performance results will hold if further updates do not increase δ_r.

C. Convergence time

Here we look at another important performance metric - algorithm convergence time, which characterizes the time it takes for the algorithm to enter the "steady state." Faster convergence implies better robustness against system statistics changes and higher efficiency in resource allocation, and is particularly important when system statistics can change. The formal definition of convergence time is as follows [17].

Definition 3. *For a given constant D, the D-convergence time of a control algorithm Π, denoted by T^{Π}_D, is the time it takes for the queue vector $(\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t))$ ($(\hat{\boldsymbol{d}}(t), \hat{\boldsymbol{Q}}(t), \hat{\boldsymbol{H}}(t))$ under* DRAM*) to get to within D distance of $\boldsymbol{\alpha}^* + \boldsymbol{\theta}$, i.e.,*

$$T^{\Pi}_D \triangleq \inf\{t\,|\,\|(\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t)) - (\boldsymbol{\alpha}^* + \boldsymbol{\theta})\| \leq D\}.\ \Diamond \tag{51}$$

This definition of convergence time concerns about when an algorithm enters its "optimal state," i.e., when the queue vectors get very close to the optimal multiplier and the algorithms start making optimal action selections. It is different from the metrics considered in [27] and [28], which concern about the time it takes for the objective value and constraints to be within certain accuracy. The following theorem summarizes the convergence performance results.

Theorem 3. *Suppose $g(\boldsymbol{\alpha})$ is polyhedral with $\rho = \Theta(1) > 0$, $\delta_\pi \leq \epsilon_z$ and $\delta_r \leq \epsilon_r$, and $\boldsymbol{\alpha}^* + \boldsymbol{\theta} \succ \mathbf{0}$. Then, with a sufficiently large V, we have:*[9]

$$\mathbb{E}\{T_{D_1}^{\texttt{LRAM}} \,|\, \mathcal{E}_r\} \;=\; O(T_{\delta_r} + \Theta(V)) \qquad (52)$$

$$\mathbb{E}\{T_{D_2}^{\texttt{DRAM}} \,|\, \mathcal{E}_r \cap \mathcal{E}_\pi\} \;=\; O((T_l + \Theta(\delta_\pi V)). \qquad (53)$$

Here $T_l \triangleq \max(T_{\delta_r}, T_{\delta_\pi})$ denotes the total learning time in `DRAM`, *$D_1 \triangleq \Theta(\delta_r V) + \Theta(1)$, and $D_2 \triangleq \Theta(\delta_r V) + \Theta(1)$. Hence, with probability P_{δ_r}, the convergence time for* `LRAM` *is $O(T_{\delta_r} + \Theta(V))$, and with probability $P_{\delta_r} P_{\delta_\pi}$, the convergence time of* `DRAM` *is $O((T_l + \Theta(\delta_\pi V))$. $\Diamond$*

Proof. See Appendix E. □

Theorem 3 shows how performance guarantees of learning modules affect the convergence time. Interestingly, `DRAM` has a convergence time of $O(\delta_\pi V)$, which can be significantly smaller than that under `DRAM` when δ_π is small. Combining Theorems 1, 2, and 3, we see that *δ_r largely affects the utility performance (reflected by D_1 and D_2), while δ_π can greatly improve the convergence time and delay.* This indicates that information of different system components can play very different roles in performance and learning accuracies should be carefully chosen for meeting a desired performance goal.

We remark here that though Theorems 2 and 3 appear to be similar to results in [17], the analysis is different due to allowing general learning methods, errors in estimating $r_n(\boldsymbol{z}(t), \mu_n(t))$, and integrating them into dual learning. Moreover, the explicit quantifications in Theorems 2 and 3 provide insight into how learning modules should be designed and how they can be integrated with control algorithms.

D. Necessity in Controlling δ_r

Here we provide a simple example showing that it is necessary to control δ_r for good utility performance. Hence, it is important to learn the reward value for each matching option. Consider the case when $N = 2$ and $M = 1$. Suppose $e(t) = A_1(t) = A_2(t) = 1$ for all t. Also suppose $\boldsymbol{b}(t) \in \{(0,1), (1,0), (0,0)\}$, that is, at every time t, we can only allocate resource to one or zero queue. Suppose $c(t) = 0$, $\mu_n(t) = b_n(t)$, and $r_n(t) = \tilde{\mu}_n(t)$. Finally, assume that $U_1(r_1) = \log(1 + r_1)$ and $\log(1 + 2r_2)$.

In this case, the true optimal takes place at $\overline{r}_1 = 1/4$ and $\overline{r}_2 = 3/4$ with $U_{\text{total}} = 0.7828$. Now suppose we incorrectly estimate the rewards to be $r_n(t) = (1 + \delta_n)\tilde{\mu}_n(t)$. Then, one can show that the optimal rewards become:

$$r_1 \;=\; \frac{2(1+\delta_1)(1+\delta_2) - 2(1+\delta_2) + (1+\delta_1)}{4(1+\delta_1)(1+\delta_2)} \qquad (54)$$

$$r_2 \;=\; \frac{2(1+\delta_1)(1+\delta_2) + 2(1+\delta_2) - (1+\delta_1)}{4(1+\delta_1)(1+\delta_2)}, \qquad (55)$$

which is roughly $r_1 \approx \frac{1}{4} + \frac{2\delta_1 - \delta_2}{4}$ and $r_2 \approx \frac{3}{4} - \frac{2\delta_1 - \delta_2}{4}$. Thus, the resulting optimal utility is given by:

$$U_{\text{total}} \approx 0.7828 - \frac{|2\delta_1| + |\delta_2|}{5}. \qquad (56)$$

Therefore, in order to obtain an $O(1/V)$ close-to-optimal utility, it is necessary to ensure that $\max(|\delta_1|, |\delta_2|) = O(1/V)$.

[9]Here the performance is obtained conditioning on the performance of the learning algorithms, i.e., $\delta_r \leq \delta$ with probability P_{δ_r} and $\delta_\pi \leq \delta$ with probability P_{δ_π}.

VIII. Simulation

In this section, we present simulation results for our algorithms. We consider a system that has $N = 2$ and $M = 1$. We assume that $e(t)$ is 0 or 2 with equal probabilities. $A_1(t)$ is 0 or 2 with equal probabilities and $A_2(t)$ is 1 or 2 with equal probabilities. $\boldsymbol{b}(t) \in \{(0,1), (1,0), (0,0)\}$, i.e., at every time t, we allocate one unit resource to one or zero queue. We set $c(t) = b_1(t) + b_2(t)$ and $\mu_n(t) = b_n(t)$, and $\gamma_n(t) \in \{0, 1, 2\}$. There are two system states $\omega(t) \in \{1, 2\}$. In each state, the reward functions are given by $r_n(t) = w_n(\omega(t))\tilde{\mu}_n(t)$, where $w_1(1) = 0.8$ and $w_2(1) = 1$ and $w_1(2) = 1$ and $w_2(2) = 0.8$. Thus, the system state indicates which tasks are preferred under specific conditions. Every time the corresponding reward is either $0.5r_n(t)$ or $1.5r_n(t)$, with equal probabilities. Finally, we assume that $U_1(r_1) = 1.2\log(1 + 2r_1)$ and that $U_2(r_2) = 1.2\log(1 + 4r_2)$.

From the definitions, we have that $\beta = 5$, $\beta_\mu^u = \beta_\mu^l = 1$ and $\beta_{\hat{r}} = \max_{w,n} \hat{r}_n(w)$. We also set $r_{\max} = 2$, $\mu_{\max} = 1$ and $b_{\max} = 1$, $h_{\max} = 2$. We simulate both `LRAM` and `DRAM` with $V = \{10, 20, 50, 80, 100\}$. According to (21) and (22), $\theta_1 = (5V + 2)\beta_{\hat{r}} + 4$ and $\theta_2 = (5V + 2)\beta_{\hat{r}} + 3$. In `DRAM`, we set $\zeta = \log(V)^2$ to see if using this smaller offset can still achieve good performance. We use $\texttt{TBS}(s_{th})$ to estimate $\boldsymbol{r}$ and set $s_{th} = \log(V)^2$, and use $\texttt{TLS}(T_{th})$ to estimate $\boldsymbol{\pi}$ and set $T_{th} = T_{tbs}$. In `LRAM`, we randomly add or subtract the estimation error δ_r from the true values.

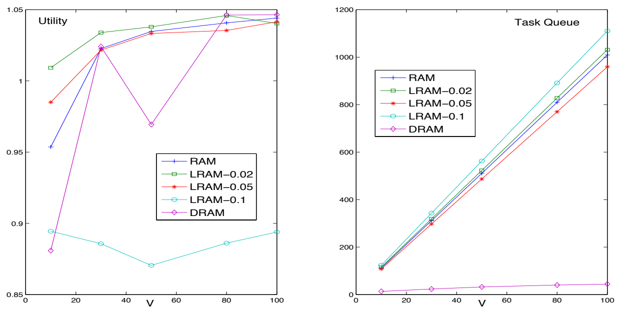


Fig. 3. Utility performance and task queue size.

Fig. 3 first shows the utility performance and the task queue behavior of `LRAM` and `DRAM`, where the number after `LRAM` denotes δ_r. We see from the left plot that except for $\delta_r = 0.1$, `LRAM` performs very well under all other error values, suggesting that estimation error indeed plays an important role in system utility. We also see that `DRAM` performs very well starting from $V \geq 50$. The right plot shows the backlog (delay) performance under different schemes. It is evident that `DRAM` achieves an $O(\log(V)^2)$ delay in this case, while all other variants possess an $O(V)$ delay. This demonstrates the importance of incorporating system dynamics information into algorithm design.

Fig. 4 then shows the resource queue $H(t)$ and deficit queues $\boldsymbol{d}(t)$. We see that `DRAM` ensures an $O(\log(V)^2)$ average resource queue, while other algorithms result in an $O(V)$ queue size. This implies that `DRAM` ensures a short stay in the system for the resource items. This feature is particularly useful if the resource items are human users, e.g., in crowdsourcing.

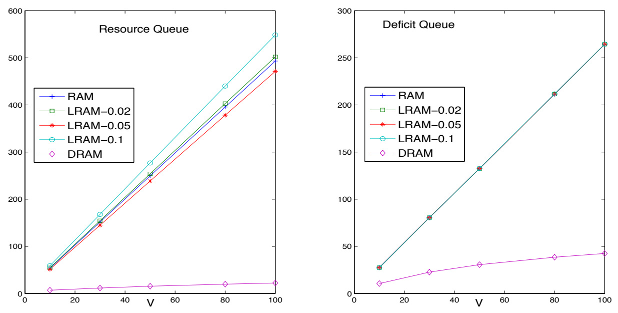


Fig. 4. Resource queue and deficit queue sizes.

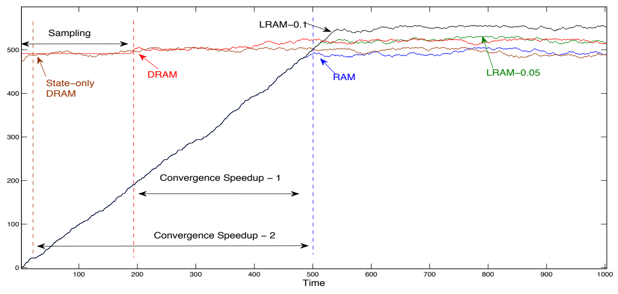


Fig. 5. Convergence of LRAM and DRAM for $V = 100$.

Finally, Fig. 5 shows the convergence behavior of the algorithms for $V = 100$. Here we show the resource queue value as its convergence time dominates the others. We see that RAM takes an $O(V)$ time to converge, which is expected. We also observe that $H(t)$ under LRAM-0.05 and LRAM-0.1 converge to values slightly above those under RAM. This explains why their performance is slightly worse. On the other hand, we also see that DRAM converges quickly. The reason its steady state value is slightly above that under RAM is due to the inaccuracy of $\hat{\boldsymbol{r}}$. Even in this case, we see that there is a $2.5\times$ convergence speedup (most of the learning time is due to sampling) and DRAM achieves very good performance. In the case when $\boldsymbol{r}$ can be obtained from other data source beforehand, which can commonly be done in practice, e.g., in online advertising, we see that DRAM (called state-only DRAM in this case) achieves a $10\times$ convergence speedup (50 slots vs. 500 slots).

We observe in all simulation instances that no dropping occurs. This demonstrates the effectiveness of the algorithms and validates Theorem 2.

IX. Conclusion

In this paper, we study the problem of optimal matching with queues in dynamic systems. We develop two online learning-aided algorithms LRAM and DRAM for resolving the challenging underflow problem and to achieve near-optimality. We show that LRAM achieves an $O(\epsilon + \delta_r)$ system utility, for any $\epsilon > 0$, while ensuring an $O(1/\epsilon)$ delay bound and an $O(1/\epsilon)$ algorithm convergence time (Here $\epsilon = 1/V$). DRAM, on the other hand, guarantees a similar $O(\epsilon + \delta_r)$ system utility, while achieving an $O(\delta_\pi/\epsilon + \log(1/\epsilon)^2)$ delay bound and an $O(\delta_\pi/\epsilon)$ algorithm convergence time, which can be significantly better compared to LRAM when δ_π is small. Our algorithms and results show that different system information can play very different roles in algorithm performance and provide insights into joint learning-control algorithm design for dynamic systems.

References

[1] N. Mckeown, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *Proceedings of INFOCOM*, 1996.

[2] A. Mehta. *Online Matching and Ad Allocation*. Foundations and Trends in Theoretical Computer Science Vol. 8, no. 4, pp. 265-368, 2013.

[3] F. Alt, A. Shirazi, A. Schmidt, U. Kramer, and Z. Nawaz. Location-based crowdsourcing: Extending crowdsourcing to the real world.

[4] Uber. https://www.uber.com/.

[5] S. Maguluri, R. Srikant, and L. Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. *Proceedings of INFOCOM*, 2012.

[6] M. J. Neely and L. Huang. Dynamic product assembly and inventory control for maximum profit. *IEEE Conference on Decision and Control (CDC), Atlanta, Georgia*, Dec. 2010.

[7] M. J. Neely. Super-fast delay tradeoffs for utility optimal fair scheduling in wireless networks. *IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Nonlinear Optimization of Communication Systems*, 24(8):1489–1501, Aug. 2006.

[8] Libin Jiang and Jean Walrand. A distributed csma algorithm for throughput and utility maximization in wireless networks. *IEEE/ACM Transactions on Networking, vol. 18, no.3, pp. 960 - 972*, Jun. 2010.

[9] C. W. Tan, D. P. Palomar, and M. Chiang. Energy-robustness tradeoff in cellular network power control. *IEEE/ACM Transactions on Networking, Vol. 17, No. 3, pp. 912-925*, 2009.

[10] X. Lin P. Huang and C. Wang. A low-complexity congestion control and scheduling algorithm for multihop wireless networks with order-optimal per-flow delay. *Proceedings of INFOCOM*, 2011.

[11] I. Hou and P.R. Kumar. Utility-optimal scheduling in time-varying wireless networks with delay constraints. *Proceedings of MobiHoc*, 2010.

[12] L. Georgiadis, M. J. Neely, and L. Tassiulas. *Resource Allocation and Cross-Layer Control in Wireless Networks*. Foundations and Trends in Networking Vol. 1, no. 1, pp. 1-144, 2006.

[13] Committee on the Analysis of Massive Data; Committee on Applied, Theoretical Statistics; Board on Mathematical Sciences, Their Applications; Division on Engineering, and Physical Sciences; National Research Council. *Frontiers in Massive Data Analysis*. 2013.

[14] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[15] T. L Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):422, 1985.

[16] H. Wu, R Srikant, X. Liu, and C. Jiang. Algorithms with logarithmic or sublinear regret for constrained contextual bandits. pages 433–441, 2015.

[17] L. Huang, X. Liu, and X. Hao. The power of online learning in stochastic network optimization. *Proceedings of ACM Sigmetrics*, 2014.

[18] L. Huang. Fast-convergent learning-aided control in energy harvesting networks. *Proceedings of IEEE CDC*, 2015.

[19] O. Simeone C. Tapparello and M. Rossi. Dynamic compression-transmission for energy-harvesting multihop networks with correlated sources. *IEEE/ACM Trans. on Networking*, 2014.

[20] S. Chen, P. Sinha, N. B. Shroff, and C. Joo. A simple asymptotically optimal joint energy allocation and routing scheme in rechargeable sensor networks. *IEEE/ACM Trans. on Networking*, to appear.

[21] B. Tan and R. Srikant. Online advertisement, optimization and stochastic networks. *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC) Orlando, FL, USA*, December 2011.

[22] L. Huang and M. J. Neely. The optimality of two prices: Maximizing revenue in a stochastic network. *IEEE/ACM Transactions on Networking*, 18(2):406–419, April 2010.

[23] L. Huang and M. J. Neely. Delay reduction via Lagrange multipliers in stochastic network optimization. *IEEE Trans. on Automatic Control*, 56(4):842–857, April 2011.

[24] T. Ji J. Ghaderi and R. Srikant. Flow-level stability of wireless networks: Separation of congestion control and scheduling.

[25] A. Eryilmaz and R. Srikant. Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control. *IEEE/ACM Trans. Netw.*, 15(6):1333–1344, 2007.

[26] T. E. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3rd edition)*. The MIT Press, 2009.

[27] B. Li, A. Eryilmaz, and R. Li. Wireless scheduling for utility maximization with optimal convergence speed. *Proceedings of IEEE INFOCOM, Turin, Italy*, April 2013.

[28] M. Neely. Energy-aware wireless scheduling with near optimal backlog and convergence time tradeoffs. *Proceedings of INFOCOM*, 2015.

[29] F. Chung and L. Lu. Concentration inequalities and martingale inequalities - a survey. *Internet Math.*, 3 (2006-2007), 79–127.

[30] S. N. Bernstein. On a modification of chebyshev?s inequality and of the error formula of laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math. 1*, 1924.

[31] L. Huang and M. J. Neely. Max-weight achieves the exact $[O(1/V), O(V)]$ utility-delay tradeoff under Markov dynamics. *arXiv:1008.0200v1*, 2010.

[32] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar. *Convex Analysis and Optimization.* Boston: Athena Scientific, 2003.

Appendix A – Proof of Lemma 1

Proof. (Lemma 1) Using the queueing dynamics (5) and (6), we have:

$$(Q_n(t+1)-\theta_1)^2 \leq (Q_n(t)-\theta_1)^2$$
$$-2(Q_n(t)-\theta_1)(\mu_n(t)-R_n(t)) + R_n(t)^2 + \mu_n(t)^2.$$

Carrying out a computation similarly for $H_m(t)$ and $d_n(t)$, and summing the above and using the definition of $L(t)$ and $\Delta_V(t)$, we get:

$$\begin{aligned} L(t+1) - L(t) - Vf_\gamma(t) &\leq G - V(\sum_n U_n(\gamma_n(t)) - c(t)) \\ &\quad - \sum_n (Q_n(t)-\theta_1)(\mu_n(t)-R_n(t)) \\ &\quad - \sum_n d_n(t)(\kappa_n(t)-\gamma_n(t)) \\ &\quad - \sum_m (H_m(t)-\theta_2)(\sum_n b_{mn}(t) - h_m(t)). \end{aligned}$$

Here $G \triangleq N(A_{\max}^2 + \mu_{\max}^2 + 2r_{\max}^2) + Mh_{\max}^2 + MN^2 b_{\max}^2$. Rearranging terms, we obtain:

$$\begin{aligned} &L(t+1) - L(t) - Vf_\gamma(t) \\ &\leq G - \sum_n [VU_n(\gamma_n(t)) - d_n(t)\gamma_n(t)] \\ &\quad + \sum_n (Q_n(t)-\theta_1)R_n(t) + \sum_m (H_m(t)-\theta_2)h_m(t) \\ &\quad + \Big[Vc(t) - \sum_m (H_m(t)-\theta_2)\sum_n b_{mn}(t) \\ &\quad\quad - \sum_n (Q_n(t)-\theta_1)\mu_n(t) - \sum_n d_n(t)\kappa_n(t)\Big]. \end{aligned}$$

Taking an expectations on both sides conditioning on $\boldsymbol{y}(t)$ and using the fact that $\kappa_n(t)$ is an i.i.d. random variable given $\boldsymbol{z}(t), \boldsymbol{b}(t), \boldsymbol{Q}(t)$, we see that the lemma follows. □

Appendix B – Proof of Lemma 2

We prove Lemma 2 here. In our proof, we use the following theorem from [29] (also known as the Bernstein inequality [30]).

Theorem 4. *[29] Suppose X_i are independent random variables satisfying $X_i \leq B$ for $1 \leq i \leq n$. Let $X = \sum_i X_i$ and $\|X\| = \sqrt{\sum_i \mathbb{E}\{X_i^2\}}$. Then we have:*

$$Pr\{X \leq \mathbb{E}\{X\} - b\} \leq e^{-\frac{b^2}{2(\|X\|^2 + Bb/3)}}.\Diamond \tag{57}$$

We now present the proof of Lemma 2.

Proof. (Lemma 2) First of all, we show that $\mathbb{E}\{T_{tbs}\} = \Theta(\log(V)^2)$. To see this, notice that since each $\mathcal{B}_k$ is finite, if the state $\boldsymbol{z}(t) = k$ appears $|\mathcal{B}_k|s_{th}$ times, we must have sampled every $\boldsymbol{b} \in \mathcal{B}_k$ s_{th} times. Thus,

$$T_{tbs} \leq \sum_k T\{\text{visit}\, \boldsymbol{z}_k\ |\mathcal{B}_k| s_{th} \text{ times}\}. \tag{58}$$

Taking the expectations, we see that $\mathbb{E}\{T_{tbs}\} \leq s_{th}\sum_k \frac{|\mathcal{B}_k|}{\pi_k}$.

Then, we see that in $\hat{\boldsymbol{r}}$, each single value has been sampled s_{th} times. Using Theorem 4 and that $\kappa_n(t) \leq r_{\max}$, we get:

$$\Pr\Big\{\sum_{t=1}^{T_{tbs}} 1_{[\boldsymbol{z}(t)=\boldsymbol{z}, \boldsymbol{b}(t)=\boldsymbol{b}]}\kappa_n(t) \leq r_n(\boldsymbol{z},\boldsymbol{b})s(\boldsymbol{z},\boldsymbol{b},T_{tbs}) - b\Big\}$$
$$\leq e^{-\frac{b^2}{2(s_{th}r_{\max}^2 + r_{\max}b/3)}}.$$

Choosing $b = \sqrt{s_{th}}\log(s_{th})$, dividing both sides of the inequality inside by $s(\boldsymbol{z},\boldsymbol{b},T_{tbs})$, and using $s(\boldsymbol{z},\boldsymbol{b},T_{tbs}) \geq s_{th}$,

$$\Pr\Big\{\hat{r}_n(\boldsymbol{z},\boldsymbol{b}) \leq r_n(\boldsymbol{z},\boldsymbol{b}) - \frac{\log(s_{th})}{\sqrt{s_{th}}}\Big\} \leq e^{-\frac{s_{th}\log(s_{th})^2}{2(r_{\max}^2 s_{th} + r_{\max}\sqrt{s_{th}}/3)}}.$$

Using Theorem 4 with $-X$, we get a similar bound for the other side. Hence,

$$\Pr\Big\{|\hat{r}_n(\boldsymbol{z},\boldsymbol{b}) - r_n(\boldsymbol{z},\boldsymbol{b})| \leq \frac{\log(s_{th})}{\sqrt{s_{th}}}\Big\} \leq 2e^{-\frac{s_{th}\log(s_{th})^2}{2(r_{\max}^2 s_{th} + r_{\max}\sqrt{s_{th}}/3)}}.$$

Using the union bound, we see that Part (a) follows. Part (b) can be proven similarly. □

Appendix C – Proof of Theorem 1

We present the proof for Theorem 1 here. For our analysis, we use the following result, which is Theorem 1 in [31]. The result states that the optimal dual value $g(\boldsymbol{\alpha}^*)$ provides an upper bound for the optimal system utility.

Theorem 5. *[31] Let $\boldsymbol{\alpha}^*$ be an optimal solution of (30). Then, $g(\boldsymbol{\alpha}^*) \geq Vf_{av}^*$.* $\Diamond$

Proof. (Theorem 1) (**Queueing**) First consider $d_n(t)$. We see that if $d_n(t) \leq V\beta$, then $d_n(t+1) \leq V\beta + r_{\max}$. On the other hand, from (13), whenever $d_n(t) > V\beta$, $\gamma_n(t) = 0$. Hence, $d_n(t)$ will not further increase. This proves the bound for $d_n(t)$. The bounds for $Q_n(t)$ and $H_m(t)$ can be similarly proven by noticing that `LRAM` will not further admit tasks once $Q_n(t) \geq \theta_1$ and it does not admit resources once $H_m(t) \geq \theta_2$.

(**Utility**) We carry out the proof by comparing the RHS of (18) under `LRAM` with any other control policy, including the ones that do not respect the no-underflow constraint (1).

To this end, consider (14). We show that even without constraint (1), `LRAM` ensures that (i) whenever $H_m(t) \leq Nb_{\max}$, $b_{mn}(t) = 0$ for all n, and (ii) whenever $Q_n(t) \leq \mu_{\max}$, $\mu_n(t) = 0$. This intermediate step allows us to compare the drift value under `LRAM` over all possible alternative policies, including ones that do not respect the no-underflow constraint. We first show (i). Suppose $H_m(t) < Nb_{\max}$. Using (22),

$$H_m(t) - \theta_2 < -(V\beta + r_{\max})\beta_{\hat{r}} - r_{\max}\beta_\mu^u. \tag{59}$$

In this case, denote the optimal resource allocation vector as $\boldsymbol{b}^*$ and suppose there is one n with $b_{mn}^* > 0$. Let $\tilde{\boldsymbol{b}}$ be the vector obtained from $\boldsymbol{b}^*$ by setting $b_{mn}^* = 0$. We have:

$$\begin{aligned} &\Psi_{\hat{r}}(\boldsymbol{b}^*) - \Psi_{\hat{r}}(\tilde{\boldsymbol{b}}) \qquad (60)\\ &= Vc(\boldsymbol{z},\boldsymbol{b}^*) - Vc(\boldsymbol{z},\tilde{\boldsymbol{b}}) - (H_m(t)-\theta_2)b_{mn}^* \\ &\quad -(Q_n(t)-\theta_1)(\mu_n(\boldsymbol{z}(t),\boldsymbol{b}^*) - \mu_n(\boldsymbol{z}(t),\tilde{\boldsymbol{b}})) \\ &\quad -d_n(t)(\hat{r}_n(\boldsymbol{z}(t),\tilde{\mu}_n(\boldsymbol{z}(t),\boldsymbol{b}^*)) - \hat{r}_n(\boldsymbol{z}(t),\tilde{\mu}_n(\boldsymbol{z}(t),\tilde{\boldsymbol{b}}))) \end{aligned}$$

$$> [(V\beta + r_{\max})\beta_{\hat{r}} + r_{\max}\beta^u_\mu]b^*_{mn} - r_{\max}\beta^u_\mu b^*_{mn} \\ -(V\beta + r_{\max})\beta_{\hat{r}} b^*_{mn} = 0.$$

In the inequality, we have used the fact that $Q_n(t) - \theta_1 \leq r_{\max}$, $\mu_n(\boldsymbol{z}(t), \boldsymbol{b}^*) - \mu_n(\boldsymbol{z}(t), \tilde{\boldsymbol{b}}) \leq \beta^u_\mu$, $d_n(t) \leq V\beta + r_{\max}$, and $\hat{r}_n(\boldsymbol{z}(t), \tilde{\mu}_n(\boldsymbol{z}(t), \boldsymbol{b}^*)) - \hat{r}_n(\boldsymbol{z}(t), \tilde{\mu}_n(\boldsymbol{z}(t), \tilde{\boldsymbol{b}})) \leq \beta_{\hat{r}} b^*_{mn}$. However, (60) contradicts with the fact that $\boldsymbol{b}^*$ is the minimizer of $\Psi_{\hat{r}}(\boldsymbol{b})$ and shows that we must have $b^*_{mn} = 0\, \forall n$ whenever $H_m(t) < Nb_{\max}$.

Now we look at the case of $\boldsymbol{Q}(t)$. Suppose $Q_n(t) < \mu_{\max}$. Then, we have:

$$Q_n(t) - \theta_1 < -(h_{\max} + (V\beta + r_{\max})\beta_{\hat{r}})/\beta^l_\mu. \tag{61}$$

We similarly let $\boldsymbol{b}^*$ be the optimal solution. Then, we construct $\tilde{\boldsymbol{b}}$ by setting $b^*_{m^*n} > 0$ to zero, where $m^* = \arg\min_m b^*_{mn}$. In this case, if $\mu_n(\boldsymbol{z}(t), \boldsymbol{b}^*) = 0$, we are done. Otherwise,

$$\Psi_{\hat{r}}(\boldsymbol{b}^*) - \Psi_{\hat{r}}(\tilde{\boldsymbol{b}}) \tag{62}$$
$$> -h_{\max} b^*_{m^*n} - (V\beta + r_{\max})\beta_{\hat{r}} b^*_{m^*n} \\ +((h_{\max} + (V\beta + r_{\max})\beta_{\hat{r}})/\beta^l_\mu)\beta^l_\mu b^*_{m^*n} = 0.$$

The inequality follows since $H_m(t) - \theta_2 \leq h_{\max}$, $d_n(t) \leq V\beta + r_{\max}$, and that $\mu_n(\boldsymbol{z}(t), \boldsymbol{b}^*) > 0$, which implies $\mu_n(\boldsymbol{z}(t), \boldsymbol{b}^*) \geq \beta^l_\mu b^*_{m^*n}$ This contradicts with the fact that $\boldsymbol{b}^*$ is the minimizer and shows that $\boldsymbol{b}^*_n = 0$.

These two properties imply that `LRAM` automatically guarantees that the no-underflow constraints are satisfied for all $H_m(t)$ and that we always have $\tilde{\mu}_n(t) = \min[Q_n(t), \mu_n(t)] = \mu_n(t)$. To compare our control policy with any other matching policies for the drift (18), we still need to show that the actions under `LRAM`, which is based on the estimated reward matrix $\hat{\boldsymbol{r}}$, ensure that the RHS of (18), defined with the true reward $\boldsymbol{r}$, is approximately minimized.

To do so, first observe that $\boldsymbol{\gamma}(t)$, $\boldsymbol{R}(t)$ and $\boldsymbol{h}(t)$ are optimally chosen given $\boldsymbol{y}(t)$ and $\boldsymbol{z}(t)$. Hence, the only approximation comes from choosing $\boldsymbol{b}(t)$. Let $\boldsymbol{b}^*_{\hat{r}}(t)$ be the chosen vector under `LRAM` and let $\boldsymbol{b}^*_r(t)$ be the vector chosen if $\boldsymbol{r}$ is used. We have:

$$\begin{aligned}\Psi_{\hat{r}}(\boldsymbol{b}^*_{\hat{r}}(t)) &\leq \Psi_{\hat{r}}(\boldsymbol{b}^*_r(t)) \qquad (63)\\ &= \Psi_r(\boldsymbol{b}^*_r(t)) + \sum_n d_n(t)[\hat{r}_n(\boldsymbol{b}^*_r(t)) - r_n(\boldsymbol{b}^*_r(t))].\end{aligned}$$

On the other hand, we also have:

$$\Psi_{\hat{r}}(\boldsymbol{b}^*_{\hat{r}}(t)) = \Psi_r(\boldsymbol{b}^*_{\hat{r}}(t)) + \sum_n d_n(t)[\hat{r}_n(\boldsymbol{b}^*_{\hat{r}}(t)) - r_n(\boldsymbol{b}^*_{\hat{r}}(t))].$$

Combining the above two equalities and using the fact that Γ_r is a $(T_{\delta_r}, P_{\delta_r}, \delta_r)$-learning module, we see that with probability P_{δ_r}, for all $t \geq T_\delta$,

$$\begin{aligned}\Psi_r(\boldsymbol{b}^*_{\hat{r}}(t)) &\leq \Psi_r(\boldsymbol{b}^*_r(t)) + 2\sum_n d_n(t)\delta_r\\ &\leq \Psi_r(\boldsymbol{b}^*_r(t)) + 2N(V\beta + r_{\max})\delta_r. \qquad (64)\end{aligned}$$

This shows that the RHS of (18) under `LRAM` is minimized to within $2N(V\beta + r_{\max})\delta_r$, over any other policies. Comparing this to $g_k(\boldsymbol{\alpha}^d, \boldsymbol{\alpha}^q, \boldsymbol{\alpha}^h)$ in (31) and using the definition of $g(\boldsymbol{\alpha}^d, \boldsymbol{\alpha}^q, \boldsymbol{\alpha}^h)$, we conclude that:

$$\begin{aligned}\Delta_V(t) &= \mathbb{E}\{L(t+1) - L(t) - Vf_\gamma(t) \mid \boldsymbol{y}(t)\}\\ &\leq G + 2N(V\beta + r_{\max})\delta_r - g(\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t))\\ &\overset{(a)}{\leq} G + 2N(V\beta + r_{\max})\delta_r - Vf^*_{\text{av}}. \qquad (65)\end{aligned}$$

Here (a) follows from Theorem 5. Taking an expectation over $\boldsymbol{y}(t)$ on both sides and carrying out a telescoping sum from $t = 0, ..., T-1$, and dividing both sides by VT, we obtain:

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{f_\gamma(t)\} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_n \mathbb{E}\{U_n(\gamma_n(t)) - c(t)\}\\ &\geq f^*_{\text{av}} - \frac{G + r_{\max}\delta_r}{V} - 2N\beta\delta_r.\end{aligned}$$

Taking a limit as $T \to \infty$, and using Jensen's inequality and the fact that $U_n(\overline{r}_n)$ is concave, we get:

$$\sum_n U_n(\overline{\gamma}_n) - \overline{c} \geq f^*_{\text{av}} - \frac{G + r_{\max}\delta_r}{V} - 2N\beta\delta_r. \tag{66}$$

Finally, using the fact that $\boldsymbol{d}(t)$ is bounded, which implies $\overline{\gamma}_n \leq \overline{r}_n$ for all n, and that $U_n(r)$ is increasing, we see that the theorem follows. □

Appendix D – Proof of Theorem 2

Here we prove the performance of `DRAM`. We carry out the analysis in the following steps. First, we show that the estimated optimal multiplier $\hat{\boldsymbol{\alpha}}^* = (\hat{\boldsymbol{\alpha}}^{d*}, \hat{\boldsymbol{\alpha}}^{q*}, \hat{\boldsymbol{\alpha}}^{h*})$, the optimal solution of $\hat{g}^{\hat{\boldsymbol{\pi}}}(\boldsymbol{\alpha})$, is close to the true optimal. Then, we show via *drift-augmentation* that the definitions of $\hat{\boldsymbol{Q}}(t)$, $\hat{\boldsymbol{H}}(t)$, and $\hat{\boldsymbol{d}}(t)$ ensure a near-optimal algorithm performance.

We now have the following lemma for the first step, which characterizes the distance between the estimated multiplier and the true value as a function of δ_π. In the lemma, we denote $\tilde{\boldsymbol{\alpha}}^*$ the optimal solution for $\tilde{g}(\boldsymbol{\alpha})$, which is $g(\boldsymbol{\alpha})$ defined with $\boldsymbol{\pi}$ and $\hat{\boldsymbol{r}}$.

Lemma 3. *Suppose $g(\boldsymbol{\alpha})$ is polyhedral with $\rho = \Theta(1) > 0$, and that $\delta_\pi \leq \epsilon_z$ and $\delta_r \leq \epsilon_r$. Then, with probability $P_{\delta_\pi}P_{\delta_r}$, we have:*

$$\|\hat{\boldsymbol{\alpha}}^* - \tilde{\boldsymbol{\alpha}}^*\| \leq \frac{2\delta_\pi V f_{\max}\vartheta}{\rho} \tag{67}$$
$$\|\boldsymbol{\alpha}^* - \tilde{\boldsymbol{\alpha}}^*\| \leq \frac{2V f_{\max}\delta_r}{\rho\eta}, \tag{68}$$

where $\vartheta \triangleq |\mathcal{Z}|(1 + r_{\max} + A_{\max} + \mu_{\max} + Nb_{\max} + h_{\max})/\eta$ and $\eta = \Theta(1)$. ◊

Proof. See Appendix F. □

In our analysis, we make use of the following two results. In particular, Lemma 4 is a result linking queue underflow probability with the difference in the average service and arrival rate, whereas Theorem 6 shows that under `LRAM`, the queue values are exponentially attracted to the optimal multipliers.

Lemma 4. *[17] Let $Q(t)$ be the size of a single queue with dynamics $Q(t+1) = [Q(t) - \mu(t)]^+ + A(t)$. Suppose $0 \leq \mu(t), A(t) \leq \mu_{\max} = \Theta(1)$ for all t and that the queue is stable. Then,*

$$\overline{\mu(t)} - \overline{A(t)} \leq \mu_{\max}\lim_{t\to\infty} Pr\{Q(t) < \mu_{\max}\}. \tag{69}$$

Here $\overline{x(t)} \triangleq \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{x(t)\}$. ◊

Theorem 6. *Under `LRAM` with reward functions $\hat{\boldsymbol{r}}$, there exist $\Theta(1)$ constants a, K, and D, such that,*

$$\mathcal{P}(D, \nu) \leq ae^{-K\nu}, \tag{70}$$

where $\mathcal{P}(D, \nu)$ is defined as:

$\mathcal{P}(D, \nu)$

$$\triangleq \lim_{t\to\infty} Pr\{\|(\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t)) - (\tilde{\boldsymbol{\alpha}}^* + \boldsymbol{\theta})\| > D + \nu\}.\diamondsuit$$

Proof. Similar to the proof of Theorem 1 in [23]. Omitted for brevity. □

Proof. (Theorem 2) To prove Theorem 2, we define the following drift-augmentation term:

$$\Delta_a(t) \triangleq -\sum_n (\hat{\alpha}_n^{q*} - \zeta)\mathbb{E}\{\mu_n(t) - R_n(t) \mid \boldsymbol{y}(t)\} \quad (71)$$
$$-\sum_n (\hat{\alpha}_n^{d*} - \zeta)\mathbb{E}\{r_n(t) - \gamma_n(t) \mid \boldsymbol{y}(t)\}$$
$$-\sum_m (\hat{\alpha}_m^{h*} - \zeta)\mathbb{E}\{\sum_n b_{mn}(t) - h_m(t) \mid \boldsymbol{y}(t)\}.$$

Adding it to both sides of (18), we get:

$$\Delta_V(t) + \Delta_a(t) \quad (72)$$
$$\leq G - V\sum_n \mathbb{E}\{U_n(\gamma_n(t)) - \hat{d}_n(t)\gamma_n(t) \mid \boldsymbol{y}(t)\}$$
$$+\sum_n (\hat{Q}_n(t) - \theta_1)\mathbb{E}\{R_n(t) \mid \boldsymbol{y}(t)\}$$
$$+\sum_m (\hat{H}_m(t) - \theta_2)\mathbb{E}\{h_m(t) \mid \boldsymbol{y}(t)\}$$
$$+\mathbb{E}\{Vc(t) - \sum_m (\hat{H}_m(t) - \theta_2)\sum_n b_{mn}(t)$$
$$-\sum_n (\hat{Q}_n(t) - \theta_1)\mu_n(t) - \sum_n \hat{d}_n(t)r_n(t) \mid \boldsymbol{y}(t)\}.$$

Note that (72) also holds under our dropping action, because it is equivalent to modifying the dynamics of $\boldsymbol{H}(t)$ to $H_m(t+1) = (H_m(t) - \sum_n b_{mn}(t))^+ + h_m(t)$. This is important, for it allows us to analyze the performance with the drift analysis.

Using Lemma 3, we know that with probability $P_{\delta_r}P_{\delta_\pi}$, $\|\tilde{\boldsymbol{\alpha}}^* - \hat{\boldsymbol{\alpha}}^*\| \leq \frac{2\delta_\pi V f_{\max}\vartheta}{\rho}$. Using $\zeta \triangleq 2\max(\delta_\pi V\log(V)^2, \log(V)^2)$, we see that when V is large, we have:

$$\frac{2\delta_\pi V f_{\max}\vartheta}{\rho} \leq \zeta/2. \quad (73)$$

Thus,

$$\tilde{\boldsymbol{\alpha}}^* - \frac{3}{2}\boldsymbol{\zeta} \leq \hat{\boldsymbol{\alpha}}^* - \boldsymbol{\zeta} \leq \tilde{\boldsymbol{\alpha}}^* - \frac{1}{2}\boldsymbol{\zeta}, \quad (74)$$

where the inequality is taken entry-wise. This implies that with probability $P_{\delta_r}P_{\delta_\pi}$, for each n,

$$\hat{d}_n(t) \leq \max(V\beta + r_{\max}, \tilde{\alpha}_n^{d*} - \zeta/2)$$
$$\leq \hat{d}_{\max} \triangleq \max(V\beta + r_{\max}, Vf_{\max}/\eta - \zeta/2).$$

The second inequality uses (87) in Appendix F. We now carry out a similar argument as in the proof of Theorem 1 and conclude that:

$$\Delta_V(t) + \Delta_a(t) \leq G - 2N\delta_r\hat{d}_{\max} - g(\hat{\boldsymbol{d}}(t), \hat{\boldsymbol{Q}}(t), \hat{\boldsymbol{H}}(t))$$
$$\leq G - 2N\delta_r\hat{d}_{\max} - Vf_{\text{av}}^*.$$

Carrying out a telescoping sum and taking a limit as in the proof of Theorem 1, we obtain:

$$\sum_n U_n(\overline{\gamma}_n) - \overline{c}$$
$$\geq f_{\text{av}}^* - \frac{G + 2N\hat{d}_{\max}\delta_r}{V} - \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{\Delta_a(t)\}.$$

It remains to show that all queues are finite. and that $\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{\Delta_a(t)\} = O(1/V)$. Then, we can conclude $\overline{r}_n \geq \overline{\gamma}_n$ and complete the proof.

To this end, we first use (70) and the definition of $\hat{\boldsymbol{d}}(t)$, $\hat{\boldsymbol{Q}}(t)$, and $\hat{\boldsymbol{H}}(t)$, to obtain that:

$$\lim_{t\to\infty} \Pr\{d_n(t) \geq \frac{3}{2}\zeta + D + \nu\} \leq ae^{-K\nu} \quad (75)$$
$$\lim_{t\to\infty} \Pr\{Q_n(t) \geq \frac{3}{2}\zeta + D + \nu\} \leq ae^{-K\nu} \quad (76)$$
$$\lim_{t\to\infty} \Pr\{H_m(t) \geq \frac{3}{2}\zeta + D + \nu\} \leq ae^{-K\nu}, \quad (77)$$

which are exactly the queueing probability bounds (48), (49), and (50). Specifically, using the definition of $\hat{\boldsymbol{d}}(t)$ and (74), we have:

$$\{d_n(t) \geq \frac{3}{2}\zeta + D + \nu\} \subset \{\hat{d}_n(t) - \tilde{\alpha}_n^{d*} \geq D + \nu\}. \quad (78)$$

This gives the bound for $d_n(t)$. The bounds for $Q_n(t)$ and $H_m(t)$ can be reasoned similarly. Using (70) again, we see that for a large V such that $\zeta \geq D + \mu_{\max} + Nb_{\max} + r_{\max} + 2\log(V)/K$,[10]

$$\lim_{t\to\infty} \Pr\{d_n(t) \leq r_{\max}\} \leq \frac{a}{V^2} \quad (79)$$
$$\lim_{t\to\infty} \Pr\{Q_n(t) \leq \mu_{\max}\} \leq \frac{a}{V^2} \quad (80)$$
$$\lim_{t\to\infty} \Pr\{H_m(t) \leq Nb_{\max}\} \leq \frac{a}{V^2}. \quad (81)$$

Combining the above bounds and Lemma 4, and that $\tilde{\boldsymbol{\alpha}}^* = O(V)$ by (87), we conclude that $\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\{\Delta_a(t)\} = O(1/V)$. Moreover, since $\boldsymbol{d}(t)$ is stable, $\overline{r}_n \geq \overline{\gamma}_n$. Finally, using (79) - (81), we see that the fraction of time dropping happens, i.e., when the claimed reward does not count, is $O(1/V^2)$. Since $\beta = \Theta(1)$, this results in an additional utility loss of $O(1/V^2)$. Hence, we conclude that:

$$\sum_n U_n(\overline{r}_n) - \overline{c} \geq f_{\text{av}}^* - \frac{G + 2N\hat{d}_{\max}\delta_r}{V} - O(1/V). \quad (82)$$

This completes the proof. □

Appendix E – Proof of Theorem 3

Proof. (Theorem 3) To prove the result, we define a different Lyapunov function as follows:

$$L_0(t) = \frac{1}{2}||(\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t)) - \boldsymbol{\theta} - \tilde{\boldsymbol{\alpha}}^*||^2. \quad (83)$$

Then, we define a one-slot conditional Lyapunov drift as $\Delta_0(t) = \mathbb{E}\{L_0(t+1) - L_0(t) \mid \boldsymbol{y}(t)\}$. Using the queueing dynamics, we obtain that:

$$\Delta_0(t) \leq G \quad (84)$$
$$-\sum_n (\tilde{\alpha}_n^{q*} - (Q_n(t) - \theta_1))\mathbb{E}\{\mu_n(t) - R_n(t) \mid \boldsymbol{y}(t)\}$$
$$-\sum_n (\tilde{\alpha}_n^{d*} - d_n(t))\mathbb{E}\{r_n(t) - \gamma_n(t) \mid \boldsymbol{y}(t)\}$$
$$-\sum_m (\tilde{\alpha}_m^{h*} - (H_m(t) - \theta_2))\mathbb{E}\{\sum_n b_{mn}(t) - h_m(t) \mid \boldsymbol{y}(t)\}.$$

Using that the last three components constitute the subgradient of $\tilde{g}(\boldsymbol{\alpha})$ at $\boldsymbol{\alpha} = (\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t))$ [32], we obtain:

$$\Delta_0(t) \leq G - (\tilde{g}((\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t)) - \boldsymbol{\theta}) - \tilde{g}(\tilde{\boldsymbol{\alpha}}^*))$$
$$\leq G - \rho||(\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t)) - \boldsymbol{\theta} - \tilde{\boldsymbol{\alpha}}^*||.$$

Therefore, for any $0 < \epsilon_0 < \rho$, if $||(\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t)) - \boldsymbol{\theta} - \tilde{\boldsymbol{\alpha}}^*|| \geq \frac{G}{\rho - \epsilon_0}$, the above implies that:

$$\mathbb{E}\{||(\boldsymbol{d}(t+1), \boldsymbol{Q}(t+1), \boldsymbol{H}(t+1)) - \boldsymbol{\theta} - \tilde{\boldsymbol{\alpha}}^*||^2 \mid \boldsymbol{y}(t)\}$$

[10]This is always possible as $\zeta = \Omega(\log(V)^2)$.

$$\leq (||(\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t)) - \boldsymbol{\theta} - \tilde{\boldsymbol{\alpha}}^*|| - \epsilon_0)^2,$$

which further implies that:

$$\mathbb{E}\big\{||(\boldsymbol{d}(t+1), \boldsymbol{Q}(t+1), \boldsymbol{H}(t+1)) - \boldsymbol{\theta} - \tilde{\boldsymbol{\alpha}}^*|| \mid \boldsymbol{y}(t)\big\} \leq ||(\boldsymbol{d}(t), \boldsymbol{Q}(t), \boldsymbol{H}(t)) - \boldsymbol{\theta} - \tilde{\boldsymbol{\alpha}}^*|| - \epsilon_0.$$

Then, using the fact that $\tilde{\boldsymbol{\alpha}}^* = \Theta(V)$ [23], $(\boldsymbol{d}(T_{\delta_r}+1), \boldsymbol{Q}(T_{\delta_r}+1), \boldsymbol{H}(T_{\delta_r}+1)) = 0$, and using Lemma 5 in [17], we conclude then:

$$\mathbb{E}\big\{\tilde{T}^{\texttt{LRAM}}_{D_1'}\big\} = \mathbb{E}\big\{T_{\delta_r} + \Theta(V)\big\}.$$

Here $D_1' = G/(\rho - \epsilon_0) = \Theta(1)$ and $\mathbb{E}\big\{\tilde{T}^{\texttt{LRAM}}_{D_1'}\big\}$ denotes the expected time to get to within D_1' of $\tilde{\boldsymbol{\alpha}}^*$. Using (68) in Lemma 3, and by defining $D_1 \triangleq D_1' + \frac{2Vf_{\max}\delta_r}{\rho\eta}$, we conclude that:

$$\mathbb{E}\big\{T^{\texttt{LRAM}}_{D_1}\big\} = \mathbb{E}\big\{T_{\delta_r} + \Theta(V)\big\}.$$

This proves (52). To prove (53), note that the main difference between DRAM and LRAM is that DRAM utilizes the system state information to "jump start" the algorithm. Using Lemma 3 again, we see that with probability $P_{\delta_\pi}P_{\delta_r}$, $||(\boldsymbol{d}(0), \boldsymbol{Q}(0), \boldsymbol{H}(0)) - \boldsymbol{\theta} - \tilde{\boldsymbol{\alpha}}^*|| \leq \frac{2\delta_\pi V f_{\max}\vartheta}{\rho}$. Combing this result and (85), we conclude that:

$$\mathbb{E}\big\{T^{\texttt{DRAM}}_{D_2}\big\} \leq \mathbb{E}\big\{T_l + \Theta(\frac{2\delta_\pi V f_{\max}\vartheta}{\rho\epsilon_0})\big\}.$$

This proves (53) and completes the proof of the theorem. □

Appendix F – Proof of Lemma 3

We first show that the optimal multipliers are bounded. Then, we use this result to bound the differences among the optimal dual values. For notation simplicity, we define $f_{\max} \triangleq N\beta r_{\max} + c_{\max}$.

Proof. (Lemma 3) Since with probability $P_{\delta_r}P_{\delta_\pi}$, $\delta_r \leq \epsilon_r$ and $\delta_\pi \leq \epsilon_z$, we have from Assumption 1 that there exists a set of actions and probabilities that guarantee (39), (40), and (41). Also, since $0 < \sum_k \hat{\pi}_k \sum_i \lambda_i^k R_{in}^k < \mathbb{E}\{A_n(t)\}$ and $0 < \sum_k \hat{\pi}_k \sum_i \lambda_i^k h_{im}^k < \mathbb{E}\{e_m(t)\}$, it can be shown that there exists $\eta_1 = \Theta(1) > 0$, such that for any subset $\mathcal{I}_n \subset \mathcal{N}$ and any subset $\mathcal{I}_m \subset \mathcal{M}$, there exist a set of actions $\{\hat{\boldsymbol{R}}_i^k\}_{k=1,\dots,|\mathcal{Z}|}^{i=1,2,\dots,\infty}$ and $\{\boldsymbol{h}_i^k\}_{k=1,\dots,|\mathcal{Z}|}^{i=1,2,\dots,\infty}$ such that:

$$\sum_k \hat{\pi}_k \sum_i \lambda_i^k \hat{R}_{in}^k = \sum_k \hat{\pi}_k \sum_i \lambda_i^k \mu_n(\boldsymbol{z}_k, \boldsymbol{b}_i^k) - \sigma_{\mathcal{I}_n}\eta_1, \quad (85)$$

where $\sigma_{\mathcal{I}_n} = 1$ if $n \in \mathcal{I}_n$ and $\sigma_{\mathcal{I}_n} = -1$ otherwise. Similarly,

$$\sum_k \hat{\pi}_k \sum_i \lambda_i^k \sum_n b_{imn}^k = \sum_k \hat{\pi}_k \sum_i \lambda_i^k h_{im}^k - \sigma_{\mathcal{I}_m}\eta_1, (86)$$

where $\sigma_{\mathcal{I}_m} = 1$ if $m \in \mathcal{I}_m$ and $\sigma_{\mathcal{I}_m} = -1$ otherwise. Then, using Lemma 1 in [17], we see that $\hat{\boldsymbol{\alpha}}^*$ obtained by solving (34) satisfies that:

$$\sum_n \hat{\alpha}_n^{d*}\eta + \sum_n |\hat{\alpha}_n^{q*}|\eta + \sum_m |\hat{\alpha}_m^{h*}|\eta \leq V f_{\max}. \quad (87)$$

Here $\eta = \min(\eta_0, \eta_1)$. Moreover, (87) also holds for $\boldsymbol{\alpha}^*$ and $\tilde{\boldsymbol{\alpha}}^*$. Now consider $\tilde{g}(\tilde{\boldsymbol{\alpha}}^*)$ and $g(\boldsymbol{\alpha}^*)$. For explanation, we write $\tilde{g}(\tilde{\boldsymbol{\alpha}}^*) = \tilde{g}(\tilde{\boldsymbol{\alpha}}^*, \tilde{\boldsymbol{\gamma}}^*, \tilde{\boldsymbol{b}}^*, \tilde{\boldsymbol{R}}^*, \tilde{\boldsymbol{h}}^*)$, where $\tilde{\boldsymbol{\gamma}}^*, \tilde{\boldsymbol{b}}^*, \tilde{\boldsymbol{R}}^*, \tilde{\boldsymbol{h}}^*$ are the optimal actions corresponding to $\tilde{\boldsymbol{\alpha}}^*$ with $\hat{\boldsymbol{r}}$ and the true distribution $\boldsymbol{\pi}$. From the definition, we know that:

$$\begin{aligned}
& g(\boldsymbol{\alpha}^*, \boldsymbol{\gamma}^*, \boldsymbol{b}^*, \boldsymbol{R}^*, \boldsymbol{h}^*) \quad (88)\\
\overset{(a)}{\geq}\ & \tilde{g}(\boldsymbol{\alpha}^*, \tilde{\boldsymbol{\gamma}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{R}}, \tilde{\boldsymbol{h}}) \\
& + \sum_k \pi_k \sum_n \alpha_n^{d*}[r_n(\boldsymbol{z}_k, \mu_n(z_k, \tilde{\boldsymbol{b}}^k)) - \hat{r}_n(\boldsymbol{z}_k, \mu_n(z_k, \tilde{\boldsymbol{b}}^k))] \\
\overset{(b)}{\geq}\ & \tilde{g}(\tilde{\boldsymbol{\alpha}}^*, \tilde{\boldsymbol{\gamma}}^*, \tilde{\boldsymbol{b}}^*, \tilde{\boldsymbol{R}}^*, \tilde{\boldsymbol{h}}^*) - V f_{\max}\delta_r/\eta \\
\overset{(c)}{\geq}\ & g(\tilde{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\gamma}}^*, \hat{\boldsymbol{b}}^*, \hat{\boldsymbol{R}}^*, \hat{\boldsymbol{h}}^*) - V f_{\max}\delta_r/\eta \\
& + \sum_k \pi_k \sum_n \tilde{\alpha}_n^{d*}[\hat{r}_n(\boldsymbol{z}_k, \mu_n(z_k, \hat{\boldsymbol{b}}^{*k})) - r_n(\boldsymbol{z}_k, \mu_n(z_k, \hat{\boldsymbol{b}}^{*k}))] \\
\geq\ & g(\tilde{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\gamma}}^*, \hat{\boldsymbol{b}}^*, \hat{\boldsymbol{R}}^*, \hat{\boldsymbol{h}}^*) - 2V f_{\max}\delta_r/\eta. \quad (89)
\end{aligned}$$

Here $\tilde{\boldsymbol{\gamma}}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{R}}, \tilde{\boldsymbol{h}}$ denote the optimal actions corresponding to $\boldsymbol{\alpha}^*$ in $\tilde{g}(\boldsymbol{\alpha})$, and (a) follows from the definition of $\tilde{g}(\boldsymbol{\alpha})$ and the fact that $g(\boldsymbol{\alpha}^*)$ achieves the supremum over all actions ($\hat{\gamma}, \tilde{\boldsymbol{b}}, \tilde{\boldsymbol{R}}, \tilde{\boldsymbol{h}}$ are for α and $\tilde{g}$). In (b), we have used that $\tilde{g}(\tilde{\boldsymbol{\alpha}}^*)$ achieves the minimum over all $\boldsymbol{\alpha}$, that the learning module guarantees that $\|\hat{\boldsymbol{r}} - \boldsymbol{r}\|_{\max} \leq \delta_r$, and (87). Finally, in (c), $\hat{\boldsymbol{\gamma}}^*, \hat{\boldsymbol{b}}^*, \hat{\boldsymbol{R}}^*, \hat{\boldsymbol{h}}^*$ are the actions corresponding to $\tilde{\boldsymbol{\alpha}}^*$ under $g(\boldsymbol{\alpha})$ and it follows again because $\tilde{g}(\tilde{\boldsymbol{\alpha}}^*)$ achieves the supremum. The last inequality follows similarly to (b). Using the polyhedral structure of $g(\boldsymbol{\alpha})$, (89) implies that:

$$\|\boldsymbol{\alpha}^* - \tilde{\boldsymbol{\alpha}}^*\| \leq \frac{2V f_{\max}\delta_r}{\rho\eta}. \quad (90)$$

This proves (68). To prove (67), note that for any $\boldsymbol{\alpha}$,

$$\tilde{g}(\boldsymbol{\alpha}) - \hat{g}(\boldsymbol{\alpha}) = \sum_k (\pi_k - \hat{\pi}_k)\tilde{g}_k(\boldsymbol{\alpha}). \quad (91)$$

Therefore, with probability P_{δ_π}, we have:

$$\begin{aligned}
& |\tilde{g}(\tilde{\boldsymbol{\alpha}}^*) - \hat{g}(\tilde{\boldsymbol{\alpha}}^*)| \\
& \leq |\mathcal{Z}|\delta_\pi\Big(V f_{\max} + \sum_n \tilde{\alpha}_n^{d*} r_{\max} + \sum_n \tilde{\alpha}_n^{q*}(A_{\max} + \mu_{\max}) \\
& + \sum_m \tilde{\alpha}_m^{h*}(N b_{\max} + h_{\max})\Big) \\
& \leq \delta_\pi V f_{\max}\vartheta,
\end{aligned}$$

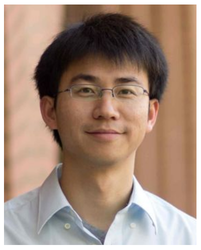
where $\vartheta \triangleq |\mathcal{Z}|(1 + r_{\max} + A_{\max} + \mu_{\max} + N b_{\max} + h_{\max})/\eta$ and the last inequality follows from (87). This implies that:

$$|\tilde{g}(\tilde{\boldsymbol{\alpha}}^*) - \tilde{g}(\hat{\boldsymbol{\alpha}}^*)| \leq 2\delta_\pi V f_{\max}\vartheta, \quad (92)$$

for otherwise we have:

$$\hat{g}(\tilde{\boldsymbol{\alpha}}^*) - \hat{g}(\hat{\boldsymbol{\alpha}}^*) \leq \tilde{g}(\tilde{\boldsymbol{\alpha}}^*) + \delta_\pi V f_{\max}\vartheta - \tilde{g}(\hat{\boldsymbol{\alpha}}^*) + \delta_\pi V f_{\max}\vartheta < 0,$$

which contradicts with the fact that $\hat{\boldsymbol{\alpha}}^*$ achieves the minimum of $\hat{g}(\boldsymbol{\alpha})$. Using the polyhedral structure, we see that (67) follows. This completes the proof of the lemma. □



Longbo Huang Longbo Huang received the Ph.D. degree in Electrical Engineering from the University of Southern California in August 2011. He then worked as a postdoctoral researcher in the Electrical Engineering and Computer Sciences department at University of California at Berkeley from July 2011 to August 2012. Since August 2012, Dr. Huang joined the Institute for Interdisciplinary Information Sciences (IIIS) at Tsinghua University (Beijing, China) as an assistant professor.

Dr. Huang was a visiting scholar at the LIDS lab at MIT in summer 2012 and summer 2014, at the EECS department at UC Berkeley in summer 2013. He was also a visiting professor at the Institute of Network Coding at CUHK in winter 2012. Dr. Huang was selected into China's Youth 1000-talent program in 2013. Dr. Huang's current research interests are in the areas of learning and optimization, network control, data center networking, smart grid and mobile networks.