A Non-Monetary Mechanism for Optimal Rate Control Through Efficient Cost Allocation

Tao Zhao, Korok Ray, and I-Hong Hou

***Abstract*— This paper proposes a practical non-monetary mechanism that induces the efficient solution to the optimal rate control problem, where each client optimizes its request arrival rate to maximize its own net utility individually, and at the Nash Equilibrium the total net utility of the system is also maximized. Existing mechanisms typically rely on monetary exchange which requires additional infrastructure that is not always available. Instead, the proposed mechanism is based on efficient cost allocation, where the cost is in terms of non-monetary metric, such as average delay or request loss rate. Specifically, we present an efficient cost allocation rule for the server to determine the target cost of each client. We then propose an intelligent policy for the server to control the costs of the clients to achieve the efficient allocation. Furthermore, we design a distributed rate control protocol with provable convergence to the Nash Equilibrium of the system. The effectiveness of our mechanism is extensively evaluated via simulations of both delay allocation and loss rate allocation against baseline mechanisms with classic control policies.**

***Index Terms*— Optimal rate control, non-monetary mechanism, efficient cost allocation, distributed protocol, state space collapse.**

I. INTRODUCTION

THE mobile Internet market has been enjoying an unprecedented growth in recent years. It is predicted that the trend will continue, and the global mobile data traffic will increase sevenfold between 2016 and 2021 [2]. With the growing market, it is of great interest to understand the economics of the network. In this paper, we are interested in finding a practical mechanism to induce the efficient solution to the optimal rate control problem in a network system of multiple selfish and strategic clients. We consider systems where a server processes requests from multiple clients, and each client can dynamically adjust its own request arrival rate. Each client obtains some utility based on its request arrival rate and its own utility function, but also suffers from some disutility based on some cost such as its experienced delay or request losses. Each client optimizes its request arrival rate to maximize its own net utility individually. The server's goal is to ensure that the total net utility is maximized at the Nash Equilibrium. Our system model can be applied to a wide range of networks. For example, the clients might be smartphones, wearable devices, tablets and so on, and the server can be a cellular base station (e.g. LTE eNodeB) or a WiFi hotspot which provides Internet services to the clients. Each request corresponds to an LTE subframe or an IP packet.

The optimal rate control problem, which entails maximizing the total net utility in the system, is typically convex, and it is thus easy to solve when one has complete information of all the individual utility functions. In practice, however, the utility functions are often private information of clients, and a strategic client that aims to maximize its own net utility may not reveal its true utility function. Further, request rates are directly controlled by clients, instead of the server. Most existing work employs some auction or pricing scheme that ensures strategic clients reveal their true functions and follow the assigned rates from the server [3], [4]. However, these schemes involve additional monetary exchange between clients and the server, which requires additional infrastructure that is not always available.

In this paper, we propose a novel non-monetary mechanism for optimal rate control to address this issue. Note that each client suffers from some disutility based on its experienced delay or request loss rate, and the server can indirectly adjust such disutility experienced by each client through its employed control policy. Therefore, the server can potentially steer request rates of strategic clients toward the optimal point through its control policy. Effectively, the server uses "delay" or "loss rate" as a kind of "currency."

In economic terms, there are negative externalities from a client increasing its request rate, since this increases the overall cost, in the form of delay or loss rate, of all clients. This is an analogy to a public goods problem [5], in which one client's consumption choice affects the utility and payoffs of the other clients. As such, the server's objective is to design an allocation scheme such that each client internalizes these negative externalities, thereby leading to efficient consumption of resources.

In designing the non-monetary mechanism, we make the following contributions:

1) First, for both the cost of delay and the cost of loss rate, we propose efficient cost allocation rules through which

Manuscript received September 1, 2017; revised January 27, 2018; accepted April 3, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor G. Paschos. This work was supported in part by NSF under Contract CNS-1719384, in part by the U.S. Army Research Laboratory and the U.S. Army Research Office under Contract/Grant W911NF-15-1-0279, in part by the Office of Naval Research under Contract N00014-18-1-2048, and in part by the Qatar National Research Fund (a member of Qatar Foundation) through NPRP under Grant 8-1531-2-651. Part of this work has been presented at WiOpt 2017 [1]. *(Corresponding author: Tao Zhao.)*

AQ:1

T. Zhao and I-H. Hou are with the Department of ECE, Texas A&M University, College Station, TX 77843 USA (e-mail: alick@tamu.edu; ihou@tamu.edu).

K. Ray is with the Mays School of Business, Texas A&M University, College Station, TX 77843 USA (e-mail: korok@tamu.edu).

Digital Object Identifier 10.1109/TNET.2018.2826457

the server can determine the cost to be allocated to each client.

2) We then design control policies used by the server to allocate costs and adjust disutilities experienced by the clients. For the cost of delay, we propose a simple scheduling algorithm and proves that it achieves the efficient delay allocation in the heavy traffic regime.[1] For the cost of loss rate, we propose a simple policy that determines which request to drop when the server's buffer is full.

3) Furthermore, we present a distributed rate control protocol where clients update their request rates based on their experienced costs. The protocol is scalable and light-weight, and is proved to converge to the Nash Equilibrium where the total net utility of the system is also maximized.

Altogether, they form our non-monetary mechanism for optimal rate control through efficient cost allocation.

The rest of the paper is organized as follows. Section II reviews the literature related to our work. Section III introduces our system model and problem formulation, using delay allocation as an example. Section IV, V, and VI present the efficient delay allocation rule, the efficient delay scheduling policy, and the distributed rate control protocol for delay allocation respectively. Section VII extends the non-monetary mechanism to loss rate allocation. Simulation study is described in Section VIII, and we conclude our paper in Section IX.

II. Related Work

There has been a considerable amount of literature that studies networks from the respect of economics. Altman et al. gave a comprehensive survey on networking games [6]. Specifically for rate control, Kelly *et al.* [4] analyzed the stability and fairness of pricing based rate control algorithms. Alpcan and Başar [7] gave a utility-based congestion control scheme for cost minimization and showed its stability for a general network topology. Hou and Kumar [3] presented a truthful and utility-optimal auction for wireless networks with per-packet deadline constraints. Gupta *et al.* [8] studied network utility maximization where flows are aggregated into flow classes. Ramaswamy *et al.* [9] considered the case when a client can choose from a number of congestion control protocols. Despite the rich literature, most existing mechanisms require additional monetary exchange between clients and the server, and infrastructure for monetary exchange is thus necessary. However, such infrastructure is not always available in wireless networks, which in turn limits the applicability of these monetary mechanisms. In contrast, our non-monetary mechanism exploits existing wireless network properties such as delay or loss rate to realize optimal rate control. The main advantage is that no additional infrastructure for monetary exchange needs to be set up or maintained, which can be a substantial cost saving.

The intellectual foundation of our research comes from economics. The early literature began with problems of creating incentives to reduce free riding in teams, such as in Groves [10]. This research uses much of the similar logic as our method on the behavior of other agents in a strategic game. Baldenius *et al.* [11], Moulin and Shenker [12], and Rajan [13] studied the problem of cost allocation, namely, how to allocate a common cost to separate corporate departments. Our contribution is combining a framework that is well utilized in economics and applying it to the optimal rate control problem in wireless networks. The application to distributed networks is new to our knowledge.

Besides, our work shares a similar spirit as the standard loss-based TCP congestion control and delay-based TCP variants, such as TCP Vegas [14], TCP Westwood+ [15], [16], and FAST TCP [17], in the sense that loss or delay is used as the signal for the clients to adjust their request rates. However, our mechanism includes not only a rate update protocol but also an efficient cost allocation rule and a control policy to enforce such rule for optimal rate control.

III. System Model for Delay Allocation

Starting from this section, we first focus on the delay allocation problem for ease of presentation. As will be shown in Section VII, the system model and mechanism design can be easily extended to the loss rate allocation problem.

Consider a system with N clients and a server. Each client i generates requests by some predefined random process, such as Poisson random process, but it can dynamically adjust its average request rate, denoted by λ_i. We use $\boldsymbol{\lambda} := [\lambda_i]$ to denote the vector containing the average request rates of all clients, and λ_{-i} to denote the vector of average request rates of all clients other than i.

On the other hand, the server employs some scheduling policy to determine which request to process. Unserved requests are queued in the system. This corresponds to real systems with sufficiently large buffers, for example, campus WiFi networks. The processing time of each request is a random variable with mean $\frac{1}{\mu}$. If the server's scheduling policy is work-conserving, which never idles as long as there is at least one request available for processing, then the average delay of all requests is a function of the total average request arrival rate, $\Lambda := \sum_i \lambda_i$, regardless of the employed scheduling policy. The average delay function $\bar{C}(\Lambda)$ is smooth, strictly increasing, and strictly convex. We assume that the average delay $\bar{C}(\Lambda)$ can be well fitted by a low-order polynomial function $C(\Lambda)$ via, for example, Chebyshev least squares approximation.

Suppose each client obtains some *utility* based on its request rate λ_i and suffers from *disutility* for every unit delay experienced by each of its request. Specifically, the utility of client i is $U_i(\lambda_i)$, where $U_i(\cdot)$ is a smooth, strictly increasing, and strictly concave function. Let $D_i(\lambda_i, \lambda_{-i})$ be the average delay that client i experiences for all its requests. The disutility of client i is $\lambda_i D_i(\lambda_i, \lambda_{-i})$. Client i aims to maximize its *net utility*, $U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \lambda_{-i})$, by choosing its request rate λ_i.

The server aims to maximize the total net utility in the system, which can be written as $\sum_i (U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \lambda_{-i}))$. Since the average delay of *all* requests is the weighted average $\frac{\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i})}{\Lambda} \approx C(\Lambda)$, we say that the server aims to

[1] Heavy traffic means the total request rate approaches the service rate.

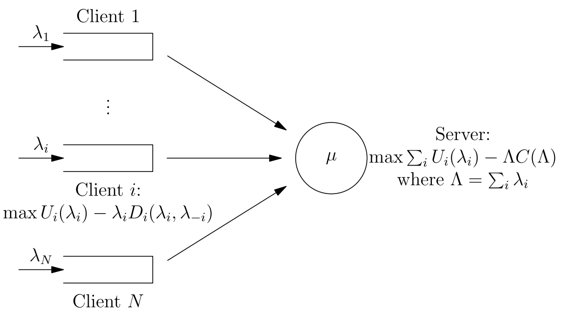


Fig. 1. An illustration of the system model.

maximize $\sum_i U_i(\lambda_i) - \Lambda C(\Lambda)$. The system model is illustrated in Fig. 1.

Note that the average delay of all requests is always infinite when the system is overloaded with $\Lambda \geq \mu$. To simplify discussions, we assume that $\boldsymbol{\lambda}$ has the properties that $\Lambda = \sum_i \lambda_i \leq (1-\epsilon)\mu$, where $\epsilon > 0$ is a predetermined value known to the server. We further assume that $\lambda_i \geq \lambda_\delta$ for all i, for some predetermined $\lambda_\delta > 0$ known to the server. These assumptions are not restrictive since we can choose ϵ and λ_δ arbitrarily close to 0. Let $\mathcal{S}_{\boldsymbol{\lambda}} := \{\boldsymbol{\lambda}\Lambda \leq (1-\epsilon)\mu, \lambda_i \geq \lambda_\delta\}$ be the feasible region of $\boldsymbol{\lambda}$. The server's optimization problem is thus formally:

$$\max_{\boldsymbol{\lambda}\in\mathcal{S}_{\boldsymbol{\lambda}}} \sum_{i=1}^{N} U_i(\lambda_i) - \Lambda C(\Lambda). \tag{1}$$

Since $U_i(\cdot)$ is concave, $C(\cdot)$ is convex, and $\mathcal{S}_{\boldsymbol{\lambda}}$ is a convex set, the problem of maximizing the total net utility can be easily solved when one has complete information of all these functions. In practice, however, the function $U_i(\cdot)$ is the private information of client i, and a strategic client may not reveal its true $U_i(\cdot)$. Now consider a game where, given $\boldsymbol{\lambda}$, the server determines the average delay experienced by each client i, $D_i(\lambda_i, \lambda_{-i})$, with the constraint that $\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i}) \geq \Lambda C(\Lambda)$. On the other hand, given λ_{-i} and $U_i(\cdot)$, each client i aims to maximize its own net utility by solving

$$\tilde{\lambda}_i = \operatorname*{argmax}_{\lambda_i} \; U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \lambda_{-i}). \tag{2}$$

Note that we allow $\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i})$ to be strictly larger than $\Lambda C(\Lambda)$, which can be achieved by employing a policy that is not work-conserving and may arbitrarily delay, or drop, requests.

We say that the system reaches a Nash Equilibrium if no client in the system can improve its own net utility unilaterally.

Definition 1: A vector $\tilde{\boldsymbol{\lambda}} := [\tilde{\lambda}_i]$ *is said to be a Nash Equilibrium if* $\tilde{\lambda}_i = \operatorname{argmax}_{\lambda_i} U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \tilde{\lambda}_{-i}), \forall i$.

Let $\boldsymbol{\lambda}^* := [\lambda_i^*]$ be the vector that maximizes the total net utility. We assume $\boldsymbol{\lambda}^*$ lies in the interior of $\mathcal{S}_{\boldsymbol{\lambda}}$ to simplify the analysis. This assumption is not restrictive by choosing ϵ and λ_δ sufficiently small. The server's problem is to find the rule that allocates delays, $[D_i(\cdot)]$, to induce optimal choices of $[\lambda_i]$.

Definition 2: A rule of allocating delays, $[D_i(\cdot)]$, *is said to be* efficient *if* $\boldsymbol{\lambda}^*$ *is the only Nash Equilibrium.*

IV. Efficient Delay Allocation

In this section, we propose the first building block of our non-monetary mechanism, an efficient delay allocation rule. The rule will be used by the server to determine how much delay should be allocated to each client given their request rates $\boldsymbol{\lambda}$.

We first study some basic properties of the optimal vector $\boldsymbol{\lambda}^* = [\lambda_i^*]$ that maximizes total net utility $\sum_i U_i(\lambda_i) - \Lambda C(\Lambda)$. We have

$$\frac{\partial}{\partial \lambda_i}\left[\sum_i U_i(\lambda_i^*) - \Lambda^* C(\Lambda^*)\right] = 0. \tag{3}$$

Hence,

$$U_i'(\lambda_i^*) = \frac{\partial}{\partial \lambda_i}\Lambda^* C(\Lambda^*). \tag{4}$$

On the other hand, if $\boldsymbol{\lambda}^*$ is also the Nash Equilibrium under some delay allocation rule $[D_i(\cdot)]$, then λ_i^* maximizes $U_i(\lambda_i) - \lambda_i D_i(\lambda_i, \lambda_{-i}^*)$, and we have

$$\frac{\partial}{\partial \lambda_i}[U_i(\lambda_i^*) - \lambda_i^* D_i(\lambda_i^*, \lambda_{-i}^*)] = 0. \tag{5}$$

Hence,

$$U_i'(\lambda_i^*) = \frac{\partial}{\partial \lambda_i}\lambda_i^* D_i(\lambda_i^*, \lambda_{-i}^*). \tag{6}$$

Combining the above equations yields

$$\frac{\partial}{\partial \lambda_i}[\Lambda^* C(\Lambda^*) - \lambda_i^* D_i(\lambda_i^*, \lambda_{-i}^*)] = 0. \tag{7}$$

Eq. (7) suggests that an efficient rule of delay allocation should ensure that $\Lambda C(\Lambda) - \lambda_i D_i(\lambda_i, \lambda_{-i})$ is only determined by λ_{-i}, and is not influenced by λ_i. It means the sum of the disutilities of all clients but i should not depend on the request rate of client i. This implication has indeed been formally stated and proved in [5]:

Proposition 1: $[D_i(\cdot)]$ *is efficient if and only if there exists functions* $R_i : \mathbb{R}^{N-1} \to \mathbb{R}$ *such that for all* i,

$$\lambda_i D_i(\lambda_i, \lambda_{-i}) = \Lambda C(\Lambda) - R_i(\lambda_{-i}), \tag{8}$$

and

$$\sum_{i=1}^{N} \lambda_i D_i(\lambda_i, \lambda_{-i}) = \Lambda C(\Lambda). \tag{9}$$

Recall that $C(\Lambda)$ is a low-order polynomial. Therefore, $\Lambda C(\Lambda)$ is also a low-order polynomial, and can be expressed as $\Lambda C(\Lambda) = c_1\Lambda + c_2\Lambda^2 + \cdots + c_m\Lambda^m$.

We now define some helpful terminology. First define the sets

$$P^j := \Big\{\boldsymbol{p} = [p_i] \;\Big|\; p_i \text{ is a nonnegative integer}, \sum_{i=1}^{N} p_i = j\Big\}, \tag{10}$$

$$P_i^j := \{\boldsymbol{p} \in P^j \big|\; p_i = 0\}, \tag{11}$$

for $j = 1, \ldots, m$ and $i = 1, \ldots, N$. Next, for $\boldsymbol{p} \in P^j$, let $G(\boldsymbol{p})$ be the number of nonzero coordinates of $\boldsymbol{p}$: $G(\boldsymbol{p}) := \big|\{l \mid p_l \neq 0\}\big|$. Note that $G(\boldsymbol{p})$ is at most j, for all $\boldsymbol{p} \in P^j$. Finally, define $\binom{j}{\boldsymbol{p}} := \frac{j!}{p_1! \cdots p_N!}$.

By the multinomial expansion theorem, it holds that

$$(\lambda_1 + \cdots + \lambda_N)^j = \sum_{\boldsymbol{p} \in P^j} \binom{j}{\boldsymbol{p}} \lambda_1^{p_1} \cdots \lambda_N^{p_N}. \tag{12}$$

We now introduce our delay allocation rule. Let

$$\beta_i^j = c_j \sum_{\boldsymbol{p} \in P_i^j} \frac{N-1}{N - G(\boldsymbol{p})} \binom{j}{\boldsymbol{p}} \lambda_1^{p_1} \cdots \lambda_N^{p_N}, \tag{13}$$

for $j = 1, \ldots, m$. We then choose $R_i(\lambda_{-i})$ as

$$R_i(\lambda_{-i}) = \sum_{j=1}^{m} \beta_i^j, \tag{14}$$

and

$$\lambda_i D_i(\lambda_i, \lambda_{-i}) = \Lambda C(\Lambda) - R_i(\lambda_{-i}). \tag{15}$$

(15) ensures that $R_i(\lambda_{-i})$ is the sum of the disutilities of all clients but i. (14) guarantees it does not depend on λ_i, which is consistent with the aforementioned implication.

Theorem 1: The rule of delay allocation $[D_i(\cdot)]$ as defined by Eq. (14) and (15) is efficient.

Proof: Since $p_i = 0$ for all $\boldsymbol{p} \in P_i^j$, it is obvious that $R_i(\lambda_{-i}) = \sum_{j=1}^m \beta_i^j$ is not influenced by λ_i.

Next, we check the condition $\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i}) = \Lambda C(\Lambda)$. By Eq. (13), for every $\boldsymbol{p} \in P^j$, the term $\frac{N-1}{N-G(\boldsymbol{p})}\binom{j}{\boldsymbol{p}}$ $\lambda_1^{p_1} \cdots \lambda_N^{p_N}$ appears in β_i^j if and only if $p_i = 0$, and there are $(N - G(\boldsymbol{p}))$ different i with $p_i = 0$. Therefore, the term $\frac{N-1}{N-G(\boldsymbol{p})}\binom{j}{\boldsymbol{p}}\lambda_1^{p_1} \cdots \lambda_N^{p_N}$ appears in $[\beta_i^j]$ a total number of $(N - G(\boldsymbol{p}))$ times. We then have

$$\begin{aligned} \sum_{i=1}^{N} R_i(\lambda_{-i}) &= \sum_{i=1}^{N} \sum_{j=1}^{m} \beta_i^j \\ &= \sum_{j=1}^{m} c_j \sum_{\boldsymbol{p} \in P^j} (N-1) \binom{j}{\boldsymbol{p}} \lambda_1^{p_1} \cdots \lambda_N^{p_N} \\ &= (N-1)\Lambda C(\Lambda), \end{aligned} \tag{16}$$

and

$$\sum_i \lambda_i D_i(\lambda_i, \lambda_{-i}) = N\Lambda C(\Lambda) - \sum_{i=1}^{N} R_i(\lambda_{-i}) = \Lambda C(\Lambda). \tag{17}$$

Therefore, by Proposition 1, the rule of delay allocation $[D_i(\cdot)]$ as defined by Eq. (14) and (15) is efficient. ∎

Next, we briefly discuss the time complexity of calculating efficient delay allocation using the above rule. The most time consuming part is obtaining all the elements of the set P^j, whose size is no more than $O(N^j)$, for all $j = 1, \ldots, m$. We can obtain P_i^j as well as $G(\boldsymbol{p})$ and $\binom{j}{\boldsymbol{p}}$ while obtaining the elements of P^j. Therefore, the total time complexity is $O(N^m)$, where m is a small constant.

Remark: We note that the allocated delays of some clients following the efficient delay allocation rule $[D_i(\cdot)]$ as in Eq. (15) might be unachievable (e.g. negative) in practice, especially when their request rates are too small compared with others. We call those clients "VIP", since their allocated delays are among the smallest. Note there is always at least one non-VIP client in the system. The above delay allocation rule is efficient only when there are no VIP clients in practical systems. In the following theoretical analysis, we will focus on the case where all clients in the system are non-VIP. We will present preliminary simulation studies on VIP clients in Section VIII-A3.

V. Efficient Delay Scheduling

In this section, we propose an online scheduling policy used by the server to ensure that the actual delay experienced by each client is the same as its allocated delay, as described in Eq. (14) and (15).

As mentioned before, we focus on non-VIP clients, and assume that $g_i := \lambda_i D_i > 0$ for all i. According to Little's law, g_i can be interpreted as the target average queue length (i.e. number of requests in the system) of client i, which is known to the server. Based on this observation, we propose the following maximum-relative-queue-length (MRQ) policy:

Definition 3 (MRQ): Let $Q_i(t)$ be the queue length of client i at time t. At time t, the MRQ policy schedules the client with the largest relative queue length, defined as $Q_i(t)/g_i$, breaking ties by scheduling the client with the lowest ID.

The intuition behind MRQ is that by always scheduling the client with the largest relative queue length, eventually all relative queue lengths are equal on average in steady state, or equivalently, the average queue length of each client is roughly the same as its target queue length.

Below we will show that the MRQ policy indeed achieves the desirable efficient delay allocation in the heavy traffic regime.[2] In particular, we show that the deviation of the actual average delay from the target delay is bounded for each client i, regardless of the difference between the total request rate Λ and the service rate μ. When Λ approaches μ, the actual average delay goes to infinity, and therefore the deviation becomes negligible compared to the actual average delay. Our technical approach is similar to the state space collapse results in the queueing theory literature [18].

Let $\boldsymbol{g} := [g_i]$ be the vector of target queue lengths for all clients. Let $\hat{\boldsymbol{g}} := \boldsymbol{g}/\sum_i g_i$ be the normalized vector of $\boldsymbol{g}$ such that $\hat{g}_i > 0$ is the fraction of target queue length for client i and $\sum_i \hat{g}_i = 1$. Define the weighted inner product of two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ by:

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle := \sum_{i=1}^{N} \frac{x_i y_i}{\hat{g}_i},$$

and the norm of a vector $\boldsymbol{x}$ by:

$$\|\boldsymbol{x}\| := \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle}.$$

[2] On the other hand, if the traffic is light and queues are not built up, it is not quite necessary to employ an advanced scheduling policy. Nevertheless, MRQ can still be used in light traffic and simulation results suggest that it works reasonably well. See also Section VIII-A2.

Note that $\|\hat{\boldsymbol{g}}\| = 1$ and thus $\hat{\boldsymbol{g}}$ is the unit vector in the direction of $\boldsymbol{g}$.

Let $\boldsymbol{Q}(t), \boldsymbol{A}(t)$, and $\boldsymbol{S}(t)$ be the vector of queue lengths, arrivals, and services respectively for all clients at time t. To simplify discussions, we assume that time is slotted and the duration of a time slot is τ. Moreover, in each time slot, each client can generate at most one request, and the server can serve at most one request. This assumption is not restrictive as we can set τ to be arbitrarily small. Next we define the generalized projection of $\boldsymbol{Q}(t)$ onto $\boldsymbol{g}$, denoted by $\boldsymbol{Q}_{\|}(t)$, as follows:

$$\boldsymbol{Q}_{\|}(t) := \langle \boldsymbol{Q}(t), \hat{\boldsymbol{g}} \rangle \hat{\boldsymbol{g}} = \sum_{i=1}^{N} Q_i(t) \hat{\boldsymbol{g}}.$$

Since the total queue length is $\sum_i Q_i(t)$, the queue length of each client i is exactly the i-th element of $\boldsymbol{Q}_{\|}(t)$ if we allocate queue lengths proportionally to $\boldsymbol{g}$. Therefore, $\boldsymbol{Q}_{\|}(t)$ can be thought of as the vector of target queue lengths of all clients under perfect state space collapse.

The deviation $\boldsymbol{Q}_{\perp}(t)$ of actual queue lengths $\boldsymbol{Q}(t)$ from the target queue lengths $\boldsymbol{Q}_{\|}(t)$ is defined as:

$$\boldsymbol{Q}_{\perp}(t) := \boldsymbol{Q}(t) - \boldsymbol{Q}_{\|}(t).$$

Now we introduce a helpful lemma to prove the state space collapse property. Our proof is based on the Lyapunov drift techniques. First, define the following Lyapunov functions:

$$V_{\perp}(t) := \|\boldsymbol{Q}_{\perp}(t)\|, \; W(t) := \|\boldsymbol{Q}(t)\|^2, \; W_{\|}(t) := \left\|\boldsymbol{Q}_{\|}(t)\right\|^2.$$

The respective drifts are defined as follows:

$$\begin{aligned} \Delta V_{\perp}(t) &:= V_{\perp}(t+\tau) - V_{\perp}(t) \\ \Delta W(t) &:= W(t+\tau) - W(t) \\ \Delta W_{\|}(t) &:= W_{\|}(t+\tau) - W_{\|}(t) \end{aligned}$$

The following lemma, adapted from [18, Lemma 7], shows that the drift $\Delta V_{\perp}(t)$ can be bounded by $\Delta W(t)$ and $\Delta W_{\|}(t)$, and absolutely bounded.

Lemma 1: We have

$$\Delta V_{\perp}(t) \leq \frac{1}{2\|\boldsymbol{Q}_{\perp}(t)\|}(\Delta W(t) - \Delta W_{\|}(t)), \tag{18}$$

and

$$|\Delta V_{\perp}(t)| \leq 2\sqrt{\frac{N}{\hat{g}_{\min}}}, \tag{19}$$

where $\hat{g}_{\min} := \min_i \hat{g}_i$.

Proof: See Appendix A. ■

Since we are considering a single server system, it is easy to see our MRQ policy stabilizes the queues of all clients as long as $\Lambda < \mu$. Therefore, $\boldsymbol{Q}(t)$ converges to a limiting random vector $\bar{\boldsymbol{Q}}$ in steady state.

Consider the following limiting queueing process: fix a vector $\hat{\boldsymbol{g}}$ of unit length with $\hat{g}_i > 0$, we consider all systems whose allocated delays satisfy $\boldsymbol{g}/\sum_i g_i = \hat{\boldsymbol{g}}$. Each system is indexed by $\varepsilon := \mu - \Lambda^{(\varepsilon)}$, where $\Lambda^{(\varepsilon)}$ is the total request arrival rate of the system. We use $\bar{\boldsymbol{Q}}^{(\varepsilon)}$ to denote the random vector of queue lengths in steady state for the system, and use $\bar{\boldsymbol{Q}}_{\perp}^{(\varepsilon)}$ to denote the deviation in steady state. The efficiency of MRQ is formally stated in the following theorem:

Theorem 2: The efficient delay allocation rule is enforced by the MRQ scheduling policy in the heavy traffic regime. That is, there exists a sequence of finite integers $\{N_r\}$ such that $\mathbb{E}\left[\left\|\bar{\boldsymbol{Q}}_{\perp}^{(\varepsilon)}\right\|^r\right] \leq N_r$ for all $r = 1, 2, \ldots$ and for all $\varepsilon > 0$.

Proof: Below the superscript $^{(\varepsilon)}$ is omitted for brevity. By [18, Lemma 1], we only need to show the Lyapunov drift $\Delta V_{\perp}(t)$ is *1)* negative when $\|\boldsymbol{Q}_{\perp}(t)\|$ is sufficiently large, and *2)* absolutely bounded. Lemma 1 has shown that *2)* is satisfied. Moreover, *1)* can be reduced to bound $\Delta W(t)$ and $\Delta W_{\|}(t)$.

Consider $\mathbb{E}[\Delta W(t) \mid \boldsymbol{Q}] := \mathbb{E}[\Delta W(t) \mid \boldsymbol{Q}(t) = \boldsymbol{Q}]$.

$$\begin{aligned} &\mathbb{E}[\Delta W(t) \mid \boldsymbol{Q}] \\ &\quad = \mathbb{E}\left[\|\boldsymbol{Q}(t+\tau)\|^2 - \|\boldsymbol{Q}(t)\|^2 \,\middle|\, \boldsymbol{Q}\right] \\ &\quad = \mathbb{E}\left[\left\|(\boldsymbol{Q}(t) + \boldsymbol{A}(t) - \boldsymbol{S}(t))^+\right\|^2 - \|\boldsymbol{Q}(t)\|^2 \,\middle|\, \boldsymbol{Q}\right] \\ &\quad \leq \mathbb{E}\left[\|\boldsymbol{Q}(t) + \boldsymbol{A}(t) - \boldsymbol{S}(t)\|^2 - \|\boldsymbol{Q}(t)\|^2 \,\middle|\, \boldsymbol{Q}\right] \\ &\quad \leq 2\mathbb{E}[\langle \boldsymbol{Q}(t), \boldsymbol{A}(t) - \boldsymbol{S}(t) \rangle \mid \boldsymbol{Q}] + K_1, \end{aligned} \tag{20}$$

where $(\cdot)^+ := \max\{0, \cdot\}$ and K_1 is a bounded constant. Below we will omit (t) in the derivation for brevity.

Given a request rate vector $\boldsymbol{\lambda}$, define a hypothetical service rate vector $\boldsymbol{\mu} := \boldsymbol{\lambda} + \varepsilon \hat{\boldsymbol{g}}$, where $\varepsilon > 0$. Note that $\mu_{\Sigma} := \sum_i \mu_i = \Lambda + \varepsilon = \mu$. Recall μ is the service rate the server can provide.

Next, we bound the term $\mathbb{E}[\langle \boldsymbol{Q}, \boldsymbol{A} - \boldsymbol{S} \rangle \mid \boldsymbol{Q}]$ in Eq. (20). Without loss of generality, suppose at time t, client 1 has the largest relative queue length, that is $Q_1(t)/g_1 \geq Q_i(t)/g_i$ for all i. Note that by the definition of the MRQ scheduling policy,

$$\langle \boldsymbol{Q}, \mathbb{E}[\boldsymbol{S} \mid \boldsymbol{Q}] \rangle = \frac{Q_1}{\hat{g}_1}\mu \geq \frac{Q_i}{\hat{g}_i}\mu.$$

Therefore,

$$\begin{aligned} &\mathbb{E}[\langle \boldsymbol{Q}, \boldsymbol{A} - \boldsymbol{S} \rangle \mid \boldsymbol{Q}] \\ &\quad = \langle \boldsymbol{Q}, \boldsymbol{\lambda} - \boldsymbol{\mu} \rangle + \langle \boldsymbol{Q}, \boldsymbol{\mu} - \mathbb{E}[\boldsymbol{S} \mid \boldsymbol{Q}] \rangle \\ &\quad = -\varepsilon \left\|\boldsymbol{Q}_{\|}\right\| - \sum_{i=1}^{N} \mu_i \left|\frac{Q_i}{\hat{g}_i} - \frac{Q_1}{\hat{g}_1}\right| \\ &\quad \leq -\varepsilon \left\|\boldsymbol{Q}_{\|}\right\| - \mu_{\min} \sum_{i=1}^{N} \left|\frac{Q_i}{\hat{g}_i} - \frac{Q_1}{\hat{g}_1}\right|, \end{aligned} \tag{21}$$

where $\mu_{\min} := \min_i \mu_i$.

Since $0 < \hat{g}_i < 1$ for all i, we know $\hat{g}_i^2 < \hat{g}_i$, and thus

$$\sum_{i=1}^{N} \left|\frac{Q_i}{\hat{g}_i} - \frac{Q_1}{\hat{g}_1}\right| \geq \sqrt{\sum_{i=1}^{N} \left(\frac{Q_i}{\hat{g}_i} - \frac{Q_1}{\hat{g}_1}\right)^2} \geq \left\|\boldsymbol{Q} - \frac{Q_1}{\hat{g}_1}\hat{\boldsymbol{g}}\right\|.$$

Further, we know $\|\boldsymbol{Q} - t\hat{\boldsymbol{g}}\| \geq \|\boldsymbol{Q}_{\perp}\|$ for all $t \in \mathbb{R}$. Hence,

$$\begin{aligned} \mathbb{E}[\langle \boldsymbol{Q}, \boldsymbol{A} - \boldsymbol{S} \rangle \mid \boldsymbol{Q}] &\leq -\varepsilon \left\|\boldsymbol{Q}_{\|}\right\| - \mu_{\min} \left\|\boldsymbol{Q} - \frac{Q_1}{\hat{g}_1}\hat{\boldsymbol{g}}\right\| \\ &\leq -\varepsilon \left\|\boldsymbol{Q}_{\|}\right\| - \mu_{\min} \|\boldsymbol{Q}_{\perp}\| \\ &\leq -\varepsilon \left\|\boldsymbol{Q}_{\|}\right\| - \delta \|\boldsymbol{Q}_{\perp}\|, \end{aligned} \tag{22}$$

for any δ such that $0 < \delta < \min_i \lambda_i$.

Substituting Eq. (22) to Eq. (20), we get

$$\mathbb{E}\left[\Delta W(t) \mid \boldsymbol{Q}\right] \leq -2\varepsilon \left\|\boldsymbol{Q}_{\parallel}\right\| - 2\delta \left\|\boldsymbol{Q}_{\perp}\right\| + K_1. \tag{23}$$

Next, we obtain a lower bound of $\Delta W_{\parallel}(t)$. Consider $\mathbb{E}\left[\Delta W_{\parallel}(t) \mid \boldsymbol{Q}\right] := \mathbb{E}\left[\Delta W_{\parallel}(t) \mid \boldsymbol{Q}(t) = \boldsymbol{Q}\right]$. Let $\boldsymbol{\Psi}(t)$ be the unused service at time t such that $\boldsymbol{Q}(t+1) = \boldsymbol{Q}(t) + \boldsymbol{A}(t) - \boldsymbol{S}(t) + \boldsymbol{\Psi}(t)$. Note that $0 \leq \psi_i \leq 1$ for all i.

$$\begin{aligned}
&\mathbb{E}\left[\Delta W_{\parallel}(t) \mid \boldsymbol{Q}\right] \\
&= \mathbb{E}\left[\left\langle \hat{\boldsymbol{g}}, \boldsymbol{Q} + \boldsymbol{A} - \boldsymbol{S} + \boldsymbol{\Psi}\right\rangle^2 - \left\langle \hat{\boldsymbol{g}}, \boldsymbol{Q}\right\rangle^2 \,\middle|\, \boldsymbol{Q}\right] \\
&= \mathbb{E}\Big[2\left\langle \hat{\boldsymbol{g}}, \boldsymbol{Q}\right\rangle \left\langle \hat{\boldsymbol{g}}, \boldsymbol{A} - \boldsymbol{S}\right\rangle + \left\langle \hat{\boldsymbol{g}}, \boldsymbol{A} - \boldsymbol{S}\right\rangle^2 \\
&\quad + 2\left\langle \hat{\boldsymbol{g}}, \boldsymbol{Q} + \boldsymbol{A} - \boldsymbol{S}\right\rangle \left\langle \hat{\boldsymbol{g}}, \boldsymbol{\Psi}\right\rangle + \left\langle \hat{\boldsymbol{g}}, \boldsymbol{\Psi}\right\rangle^2 \,\Big|\, \boldsymbol{Q}\Big] \\
&\geq 2\left\langle \hat{\boldsymbol{g}}, \boldsymbol{Q}\right\rangle \left\langle \hat{\boldsymbol{g}}, \boldsymbol{\lambda} - \mathbb{E}\left[\boldsymbol{S} \mid \boldsymbol{Q}\right]\right\rangle \\
&\quad - 2\mathbb{E}\left[\left\langle \hat{\boldsymbol{g}}, \boldsymbol{S}\right\rangle \left\langle \hat{\boldsymbol{g}}, \boldsymbol{\Psi}\right\rangle \mid \boldsymbol{Q}\right] \\
&\geq 2\left\langle \hat{\boldsymbol{g}}, \boldsymbol{Q}\right\rangle \left\langle \hat{\boldsymbol{g}}, \boldsymbol{\lambda} - \mathbb{E}\left[\boldsymbol{S} \mid \boldsymbol{Q}\right]\right\rangle - K_2,
\end{aligned} \tag{24}$$

where $K_2 := 2\ N^2$ considering $S_i \leq 1$ and $\psi_i \leq 1$ for all i. The first term can be further reduced as follows:

$$2\left\langle \hat{\boldsymbol{g}}, \boldsymbol{Q}\right\rangle \left\langle \hat{\boldsymbol{g}}, \boldsymbol{\lambda} - \mathbb{E}\left[\boldsymbol{S} \mid \boldsymbol{Q}\right]\right\rangle = 2\left\|\boldsymbol{Q}_{\parallel}\right\| (\Lambda - \mu) = -2\varepsilon \left\|\boldsymbol{Q}_{\parallel}\right\|.$$

Therefore,

$$\mathbb{E}\left[\Delta W_{\parallel}(t) \mid \boldsymbol{Q}\right] \geq -2\varepsilon \left\|\boldsymbol{Q}_{\parallel}\right\| - K_2. \tag{25}$$

By taking expectation of Eq. (18), and substituting Eq. (23) and (25) into it, we have

$$\mathbb{E}\left[\Delta V_{\perp}(t) \mid \boldsymbol{Q}\right] \leq -\delta + \frac{K_1 + K_2}{2\left\|\boldsymbol{Q}_{\perp}\right\|},$$

which establishes the negative drift of $\mathbb{E}\left[\Delta V_{\perp}(t) \mid \boldsymbol{Q}\right]$. Along with the absolute boundness provided by Lemma 1, we can conclude that the conditions for [18, Lemma 1] are satisfied, and thus there exists a sequence of finite integers $\{N_r\}$ such that $\mathbb{E}\left[\left\|\bar{\boldsymbol{Q}}_{\perp}^{(\varepsilon)}\right\|^r\right] \leq N_r$ for all $r = 1, 2, \ldots$. ∎

Remark: Since the constants in these bounds are all independent of ε, the deviation of the limiting queue length vector $\bar{\boldsymbol{Q}}^{(\varepsilon)}$ from the target queue length vector $\boldsymbol{g}$ becomes negligible as $\varepsilon \to 0$. Therefore, we observe the state space collapse behavior of relative queue lengths, and the efficient delay allocation rule is enforced by our MRQ scheduling policy in the heavy traffic regime.

VI. Distributed Rate Control Protocol

Theorem 1 has shown that our proposed delay allocation rule in Section IV is efficient. That is, suppose there is a unique vector $\boldsymbol{\lambda}^* = [\lambda_i^*]$ that maximizes total net utility $\sum_i U_i(\lambda_i) - \Lambda C(\Lambda)$ in Eq. (1), then $\boldsymbol{\lambda}^*$ is also the unique vector of Nash Equilibrium under our delay allocation rule. Theorem 2 further proves that our MRQ scheduling policy enforces the delay allocation rule, that is each client experiences its own allocated delay in the heavy traffic regime. In this section, we propose a distributed rate control protocol for clients to dynamically adjust their rates so as to converge to the Nash Equilibrium.

Our protocol is based on the projected gradient method [19], a simple yet effective method to solve convex optimization problems. The projected gradient method consists of two steps: initialization and iterative update. In the initialization step, the method arbitrarily chooses a vector $\boldsymbol{\lambda}(0) \in \mathcal{S}_{\boldsymbol{\lambda}}$. Recall that $\mathcal{S}_{\boldsymbol{\lambda}}$ is the feasible region for $\boldsymbol{\lambda}$. In each subsequent iteration k, the projected gradient method updates $\boldsymbol{\lambda}$ by:

$$\hat{\boldsymbol{\lambda}}(k+1) = \boldsymbol{\lambda}(k) + \kappa(k)\nabla\left[\sum_{i=1}^{N} U_i(\lambda_i) - \Lambda C(\Lambda)\right],$$

$$\boldsymbol{\lambda}(k+1) = \mathrm{P}(\hat{\boldsymbol{\lambda}}(k+1)),$$

where $\kappa(k) > 0$ is the step size at the k-th iteration, and P is the projection to the convex set $\mathcal{S}_{\boldsymbol{\lambda}}$. Note that the index k of iteration should not be confused with the time slot for scheduling. We assume a *time scale separation*, where rate update happens in a more coarse time scale than scheduling, so that there is sufficient time for the scheduling policy to steer the clients and enforce the efficient delay allocation rule. Reference [19] has shown that the projected gradient method converges to the unique optimal solution, and therefore also converges to the Nash Equilibrium.

Proposition 2: If $\kappa(k)$ satisfies $\sum_{k=0}^{\infty} \kappa(k) = \infty$ and $\sum_{k=0}^{\infty} \kappa^2(k) < \infty$, then the projected gradient method either stops at some iteration k, or the infinite sequence $\{\boldsymbol{\lambda}(k)\}$ generated by the method converges to the optimal point.

Note that stopping at some iteration k means the method reaches the optimality in finite steps. However, the projected gradient method is a centralized algorithm. In particular, calculating the projection $\boldsymbol{\lambda}(k+1) = \mathrm{P}(\hat{\boldsymbol{\lambda}}(k+1))$ requires the knowledge of all elements in $\hat{\boldsymbol{\lambda}}(k+1)$. Below, we propose a distributed rate control protocol that is inspired by the projected gradient method.

Since

$$\frac{\partial}{\partial \lambda_i}[\Lambda C(\Lambda)] = \frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}\frac{\partial \Lambda}{\partial \lambda_i} = \frac{\mathrm{d}}{\mathrm{d}\Lambda}[\Lambda C(\Lambda)],$$

$\hat{\boldsymbol{\lambda}}(k+1)$ can be acquired by each client updating its own request rate:

$$\hat{\lambda}_i(k+1) = \lambda_i(k) + \kappa(k)\left[U_i'(\lambda_i(k)) - \frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}\right].$$

Note that, to facilitate the update, the server only needs to broadcast the value of $\kappa(k)$ and $\frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}$ in each iteration to all clients.

To ensure that $\boldsymbol{\lambda}(k+1)$ satisfies $\Lambda(k+1) \leq (1-\epsilon)\mu$ and $\lambda_i(k+1) \geq \lambda_\delta$, each client i further chooses

$$\lambda_i(k+1) = \min\{\max\{\hat{\lambda}_i(k+1), \lambda_\delta\}, \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}\}.$$

This step ensures that $\lambda_\delta \leq \lambda_i(k+1) \leq \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$, and therefore $\Lambda(k+1) \leq \Lambda(k)\frac{(1-\epsilon)\mu}{\Lambda(k)} = (1-\epsilon)\mu$. We also note that, to facilitate this step, the server only needs to broadcast the value of $\Lambda(k)$ in each iteration. Fig. 2 illustrates the different projection behaviors of the centralized projected gradient method and our distributed rate control protocol. Note that distributed projection requires the constraints of the optimization problem are either decoupled for each client or in a summation form, while centralized projection works with a general convex set as the feasible region.

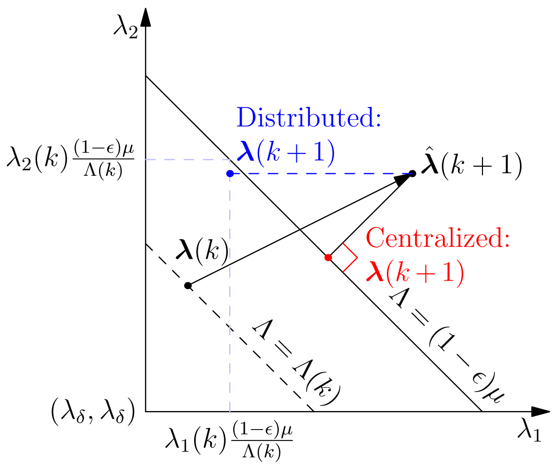


Fig. 2. Centralized vs. distributed projection.

The complete distributed protocol is summarized in Protocol 1. Compared with the centralized method, our distributed protocol is more scalable and lightweight, since it utilizes the broadcast nature of wireless channel and requires less resource of the server and the channel.

> *Protocol 1 (Distributed Rate Control Protocol):*
>
> **Server:** on convergence of relative queue lengths:
> 1. $k \leftarrow k+1$
> 2. Broadcast $\Lambda(k), \kappa(k)$, and $\frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}$
>
> **Client** i**:** on reception of server broadcast message:
> 1. Update: $\hat{\lambda}_i \leftarrow \lambda_i + \kappa(k)\left[U_i'(\lambda_i) - \frac{\mathrm{d}[\Lambda C(\Lambda)]}{\mathrm{d}\Lambda}\right]$
> 2. Projection: $\lambda_i \leftarrow \min\{\max\{\hat{\lambda}_i, \lambda_\delta\}, \lambda_i\frac{(1-\epsilon)\mu}{\Lambda}\}$

We can prove that our distributed protocol also converges to the Nash Equilibrium. This property will also be verified by simulations in Section VIII.

Theorem 3: If $\kappa(k)$ *satisfies* $\sum_{k=0}^{\infty}\kappa(k) = \infty$ *and* $\sum_{k=0}^{\infty}\kappa^2(k) < \infty$*, then the distributed rate control protocol either stops at some iteration* k*, or the infinite sequence* $\{\boldsymbol{\lambda}(k)\}$ *generated by the protocol converges to the Nash Equilibrium of the system.*

Proof: See Appendix B. ∎

VII. Non-Monetary Protocol With Efficient Loss Rate Allocation

Our non-monetary mechanism can be extended to deal with different costs other than delay itself. In this section, we consider loss rate allocation in a finite-buffer system as an example. This is more practical for real systems with only small buffers where packet losses are more common, for example, mobile hotspots set up by cellphones.

A. System Model for the Loss Rate Allocation Problem

Similar to Section III, suppose that there are N clients and a server in the system. Each client i controls its request arrival rate λ_i, and the service time needed by each request is a sequence of i.i.d. random variables with mean $\frac{1}{\mu}$. On the other hand, we assume that the server serves all requests in a first-in-first-out (FIFO) fashion, and that the server only has a finite buffer that can hold B unfinished requests, including the one being served. When the buffer is full and there is another request arrival, the server needs to drop a request to accommodate the new request, and the corresponding client experiences a loss.[3]

Since the service times of all requests have the same probability distribution, the request loss rate, defined as the average number of dropped requests per unit time, is a function of total request arrival rate, $\Lambda = \sum_i \lambda_i$. We denote the request loss rate by $\bar{L}(\Lambda)$, and note that $\bar{L}(\Lambda) = \Lambda P_B(\Lambda)$, where $P_B(\Lambda)$ is the blocking probability of the queueing system. We assume that $\bar{L}(\Lambda)$ can be well fitted by a low-order polynomial function $L(\Lambda)$, which is strictly increasing and strictly convex.

Each client obtains some utility $U_i(\lambda_i)$ based on its own request rate, and suffers from some disutility that equals its own loss rate. We use $l_i(\lambda_i, \lambda_{-i})$ to denote the loss rate of client i. Hence, the net utility of client i is $U_i(\lambda_i) - l_i(\lambda_i, \lambda_{-i})$.

Obviously, we have $\sum_i l_i(\lambda_i, \lambda_{-i}) = \bar{L}(\Lambda) \approx L(\Lambda)$. The goal of the server is to maximize the total net utility in the system, which can be approximated by $\sum_i U_i(\lambda_i) - L(\Lambda)$, while each client i aims to maximize its own net utility $U_i(\lambda_i) - l_i(\lambda_i, \lambda_{-i})$. The server can allocate the loss rate $l_i(\lambda_i, \lambda_{-i})$ of each client i through its policy of dropping requests, subject to the constraint that $\sum_i l_i(\lambda_i, \lambda_{-i}) = L(\Lambda)$.

Similar to delay allocation, we can define Nash Equilibrium and efficient allocation rule for loss rate allocation as follows:

Definition 4: A vector $\tilde{\boldsymbol{\lambda}} := [\tilde{\lambda}_i]$ *is said to be a Nash Equilibrium for loss rate allocation if* $\tilde{\lambda}_i = \operatorname{argmax}_{\lambda_i} U_i(\lambda_i) - l_i(\lambda_i, \tilde{\lambda}_{-i}), \forall i$.

Definition 5: A rule of allocating loss rates, $[l_i(\cdot)]$*, is said to be* efficient *if* $\boldsymbol{\lambda}^*$ *is the only Nash Equilibrium.*

B. Mechanism Design for Efficient Loss Rate Allocation

Our results of efficient allocation rule in Section IV can be easily extended to loss rate allocation. In particular, we have the following proposition:

Proposition 3: $[l_i(\cdot)]$ *is efficient if and only if there exists functions* $R_i : \mathbb{R}^{N-1} \to \mathbb{R}$ *such that for all* i*,*

$$l_i(\lambda_i, \lambda_{-i}) = L(\Lambda) - R_i(\lambda_{-i}), \tag{26}$$

and

$$\sum_{i=1}^{N} l_i(\lambda_i, \lambda_{-i}) = L(\Lambda). \tag{27}$$

For the allocation rule, redefine c_i to be the coefficients of $L(\Lambda)$ instead of $\Lambda C(\Lambda)$ in Section IV. Then setting

$$l_i(\lambda_i, \lambda_{-i}) = L(\Lambda) - R_i(\lambda_{-i}), \tag{28}$$

is efficient, where $R_i(\lambda_{-i})$ has the same form as in (14).

Next, we discuss how to design a policy that ensures the actual perceived loss rate of each client i is close to the

[3] The dropped request can be the newly arriving one, or some request already in the buffer.

desirable $l_i(\lambda_i, \lambda_{-i})$. Suppose at time t, the server's buffer is full and one more client request arrives. Let $\bar{l}_i(t)$ be the perceived loss rate of client i till time t for all i. On the other hand, l_i is the allocated loss rate according to the above allocation rule. We propose the following drop-smallest-relative-loss-rate (DropSRLR) policy:

Definition 6 (DropSRLR): Suppose the server's buffer is full and a new request arrives at time t, the DropSRLR policy drops a request from the client with the smallest relative loss rate, defined as $\bar{l}_i(t)/l_i$, breaking ties by choosing the client with the lowest ID.

The intuition of our dropping policy is that by always selecting the client with the smallest relative loss rate, over a long term all relative loss rates tend to be the same, which is equivalent to say each client obtains a loss rate as allocated. The efficiency of the policy will be demonstrated in the simulations in Section VIII.

Moreover, we can extend our distributed rate control protocol to loss rate allocation. The complete distributed protocol is summarized in Protocol 2. Note that there is no upper limit for the total request rate to make the finite-buffer system stable. Therefore, the distributed protocol is essentially the same as its centralized counterpart, and its convergence is straightforward to show.

> *Protocol 2 (Distributed Rate Control Protocol for Loss Rate Allocation):*
>
> **Server:** on convergence of relative loss rates:
> 1. $k \leftarrow k + 1$
> 2. Broadcast $\kappa(k)$ and $L'(\Lambda(k))$
>
> **Client** i**:** on reception of server broadcast message:
> 1. Update: $\hat{\lambda}_i \leftarrow \lambda_i + \kappa(k)\,[U_i'(\lambda_i) - L'(\Lambda(k))]$
> 2. Projection: $\lambda_i \leftarrow \max\{\hat{\lambda}_i, \lambda_\delta\}$

VIII. Simulations

In this section, we evaluate the performance of our overall design via simulations. We will present the simulations for delay allocation and loss rate allocation respectively.

A. Simulations of Delay Allocation

For delay allocation, we validate the polynomial approximation assumption for the average delay function, the state space collapse behavior of relative queue lengths through the MRQ scheduling policy, and the convergence to the Nash Equilibrium of our distributed rate control protocol. For comparison, we also consider a baseline mechanism with the classic FIFO policy for scheduling and centralized projected gradient method for rate control. Note that with FIFO scheduling, each client experiences the same average delay, i.e. $D_i(\lambda_i, \lambda_{-i}) = C(\Lambda)$.

In our simulations, we consider two systems each with $N = 10$ clients and one server. Both systems have Poisson arrivals of requests from all clients. The service time distribution of one system is exponential, and the other is deterministic. Hence, the two systems correspond to an M/M/1 queue

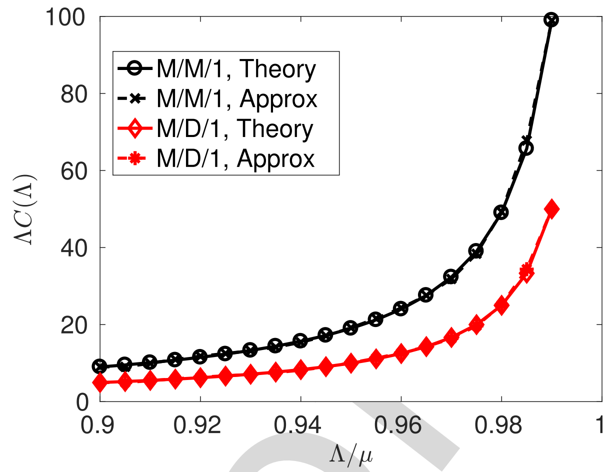


Fig. 3. Polynomial approximation of total disutility functions $\Lambda C(\Lambda)$.

and an M/D/1 queue respectively. Each system has an average service rate $\mu = 1 \times 10^3 \mathrm{s}^{-1}$ and an initial total average request rate $\Lambda = 0.95\mu = 0.95 \times 10^3 \mathrm{s}^{-1}$. Request rates will be updated by clients over time. We round up all inter-arrival times between two consecutive requests and service times of requests to the nearest microsecond. Given the above average service rate, we make about 10^3 scheduling decisions every second.

1) Polynomial Approximation of Average Delay Function: First, we evaluated the assumption that the average delay function can be well approximated by a polynomial $C(\Lambda)$. There are two methods to obtain the average delay function: One is via the theoretical formula, and the other is via simulations. Here, we use the first method. For the M/M/1 queue, the theoretical average delay function is:

$$\bar{C}(\Lambda) = \frac{1}{\mu - \Lambda}.$$

For the M/D/1 queue, it is:

$$\bar{C}(\Lambda) = \frac{1}{\mu} + \frac{\Lambda}{2\mu(\mu - \Lambda)}.$$

In our simulations, we fit $\bar{C}(\Lambda)$ with ten samples in our most interested heavy traffic region, where $\Lambda/\mu \in [0.9, 0.99]$, to get the polynomial $C(\Lambda)$. Recall that the total disutility in terms of total average queue length is $\Lambda C(\Lambda)$. The total disutility functions before and after approximation are compared in Fig. 3, labeled as "Theory" and "Approx" respectively. We can observe that the polynomial approximation fits the theoretical functions very well. In fact, the order of the polynomial $C(\Lambda)$ is as small as six, and the largest relative error of the approximation is only about 2.66%.

2) Scheduling Policy: We implemented our MRQ scheduling policy and validated the state space collapse behavior in the simulations. We use a new metric, *the relative difference of queue lengths*, defined as:

$$\left(\max_i \frac{Q_i(t)}{g_i} - \min_i \frac{Q_i(t)}{g_i}\right) \Big/ \sum_i \frac{Q_i(t)}{g_i}$$

to evaluate the state space collapse performance. Theorem 2 has shown that, given the target queue length g_i of each

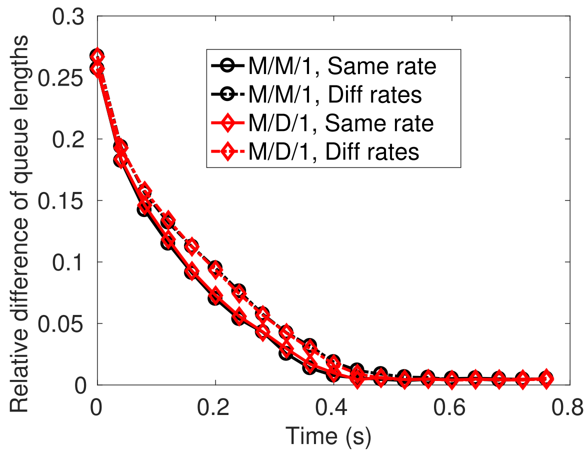


Fig. 4. State space collapse of relative queue lengths.

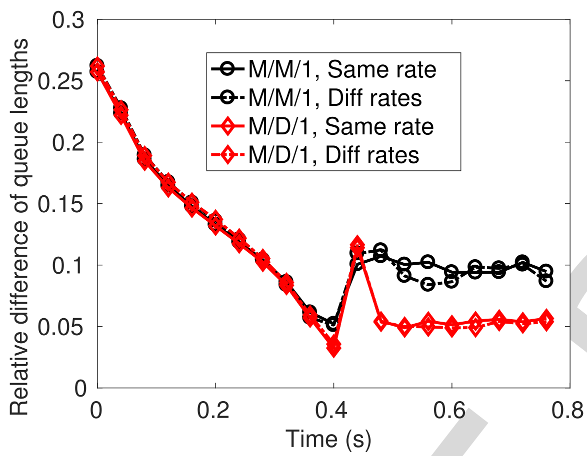
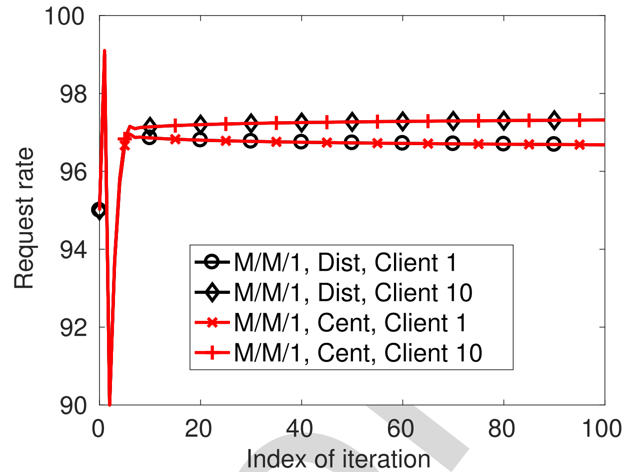
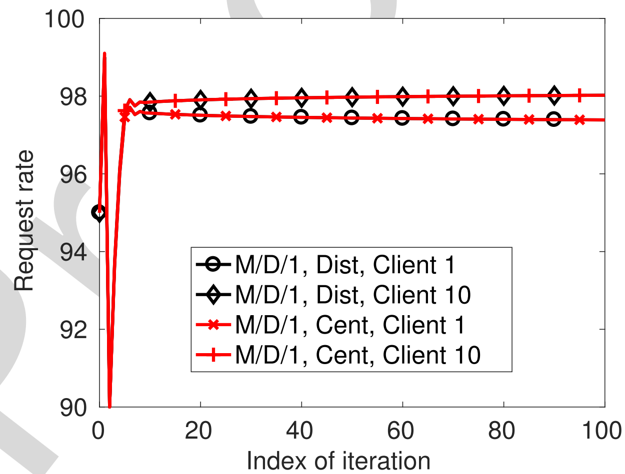


Fig. 5. State space collapse in light traffic.



(a)



(b)

Fig. 6. Convergence performance of request rates for delay allocation. (a) M/M/1 system. (b) M/D/1 system.

client i, our MRQ policy ensures that the relative difference of queue lengths converges to 0 in the heavy traffic regime.

Fig. 4 shows the evolution of the relative difference of queue lengths for both systems for two sets of initial request rates, "Same rate" and "Diff rates". "Same rate" means all ten clients have the same request rate $\lambda = \Lambda/N = 95\mathrm{s}^{-1}$, while in "Diff rates" we have two groups of request rates: $\lambda_i = 95.6\mathrm{s}^{-1}$ for $i = 1, 2, \ldots, 5$ and $94.4\mathrm{s}^{-1}$ for $i = 6, 7, \ldots, 10$. We initialize the queue length of client i to be i^2 to exhibit the convergence of relative queue lengths more clearly. We can see that the relative difference of queue lengths converges to 0 quickly for each scenario.

Fig. 5 depicts the state space collapse behavior in light traffic, where the total load of each system is only 0.1. Here "Same rate" means all ten clients have the same request rate $\lambda = \Lambda/N = 10\mathrm{s}^{-1}$, while in "Diff rates" the two groups of request rates are: $\lambda_i = 10.6\mathrm{s}^{-1}$ for $i = 1, 2, \ldots, 5$ and $9.4\mathrm{s}^{-1}$ for $i = 6, 7, \ldots, 10$. Similar to Fig. 4, the queue length of client i is initialized to be i^2. We observe that the relative difference of queue lengths also quickly decreases to a low level initially where there are enough requests to schedule. However, there is no further decrease afterwards since too few requests are in the system to achieve exact allocation of queue lengths and delays.

3) Nash Equilibrium: Furthermore, we evaluated our distributed rate control protocol in the simulations. We set the utility functions for both systems to be $U_i(\lambda_i) = \alpha w_i \log \lambda_i$, where $\alpha = 100$ is the common scaling coefficient for all clients, and w_i's are different weights for different clients. We set the weights to be in two groups: $w_i = 0.99$ for $i = 1, 2, \ldots, 5$ and 1.01 for $i = 6, 7, \ldots, 10$. Therefore, the evolution of request rates of all the clients can be captured by those of Client 1 and Client 10. For the step size, we let $\kappa(k) = 10/k$ for all k.

Fig. 6 shows the rate convergence performance for the two systems respectively. We can see that for each system, the request rates converge to two distinct values after tens of iterations. Observe that the distributed rate control protocol ("Dist" in the figure) has almost the same rate updates as the projected centralized gradient method ("Cent" in the figure). It validates that the request rates converge to the optimal rates $\boldsymbol{\lambda}^*$, and the distributed rate control protocol achieves the Nash Equilibrium of the system.

Fig. 7 shows the convergence performance in terms of total net utility for the two systems. The total net utility settles down quickly with our distributed protocol ("MRQ, Dist" in

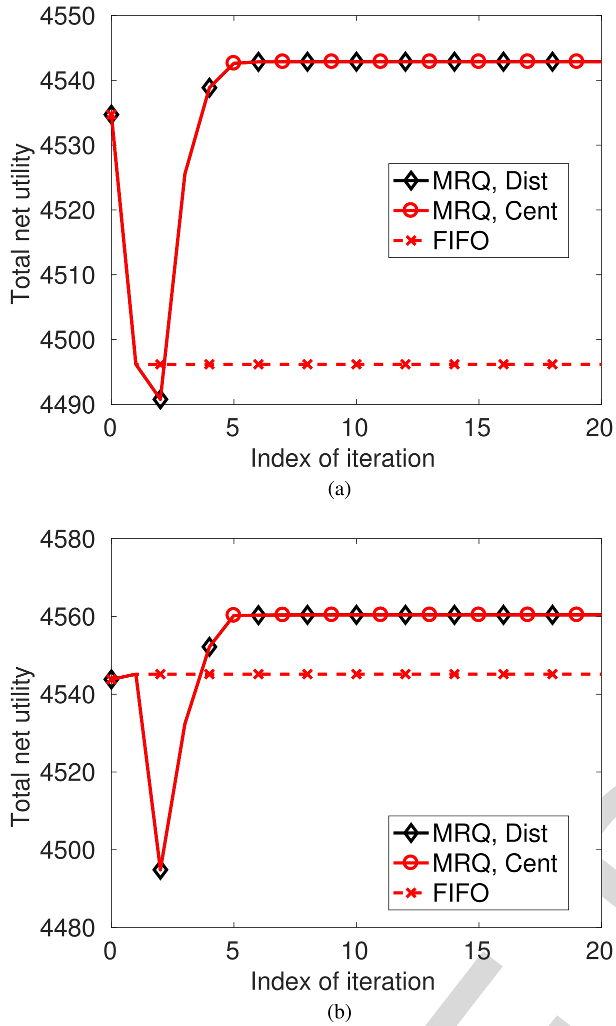


Fig. 7. Convergence performance of total net utility for delay allocation. (a) M/M/1 system. (b) M/D/1 system.

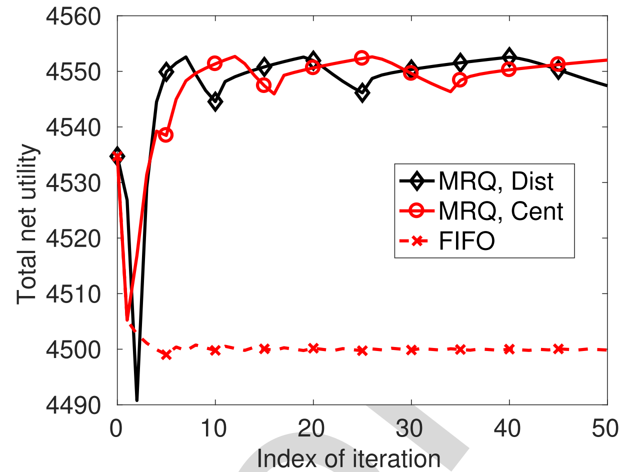


Fig. 8. Total net utility evolution with VIP clients at the Nash Equilibrium.

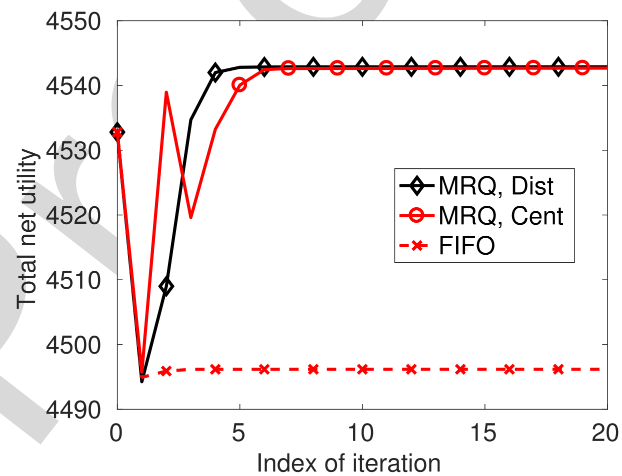


Fig. 9. Total net utility convergence with VIP clients at initial arrival.

the figures), and the evolution is again almost the same as the centralized method ("MRQ, Cent" in the figures). It means the total net utility converges to the optimal value of the optimization problem, and confirms the convergence of our distributed rate control protocol. In these figures we also plot the performance of the baseline mechanism with the FIFO scheduling policy for comparison. We can see that under the baseline mechanism, the total net utility converges to a suboptimal value. It indicates that the delay allocation rule of the baseline mechanism is not efficient.

We also conduct preliminary studies on the impact of VIP clients via simulations. We assume VIP clients experience zero delay and update their request rates accordingly. We consider two scenarios where VIP clients exist. One is that there are VIP clients at the Nash Equilibrium. In the simulation, we set the weights in the utility functions to be $w_i = 0.7$ for $i = 1, 2, \ldots, 5$ and 1.3 for $i = 6, 7, \ldots, 10$ so that Clients 1–5 will be VIP at the Nash Equilibrium. Fig. 8 depicts the evolution of total net utility for the M/M/1 system in this scenario. Observe that under our protocol, the total net utility oscillates over time. However, our mechanism still outperforms the baseline mechanism. The other scenario uses the same utility functions as those in Fig. 7, but sets the initial request rates to be $\lambda_i = 100\text{s}^{-1}$ for $i = 1, 2, \ldots, 5$ and 90s^{-1} for $i = 6, 7, \ldots, 10$ so that initially Clients 6–10 are VIP. We find that Clients 6–10 remain VIP clients in the first two iterations. However, all clients are non-VIP afterwards. Fig. 9 shows the convergence performance of total net utility for the M/M/1 system. Note that it converges to the same optimal value as in Fig. 7a under our mechanism. Therefore, in this case the system eventually converges to the original optimal Nash Equilibrium.

B. Simulations of Loss Rate Allocation

For loss rate allocation, we will show the validity of polynomial approximation for the loss rate function, the convergence of relative loss rates under our DropSRLR policy, and the convergence of the distributed rate control protocol in Protocol 2. As for the baseline mechanism, we use the well-known DropTail policy that always drops the newly arriving request when the buffer is full. Note that under DropTail, each client has the same blocking probability, and thus $l_i(\lambda_i, \lambda_{-i}) = \lambda_i L(\Lambda)/\Lambda$.

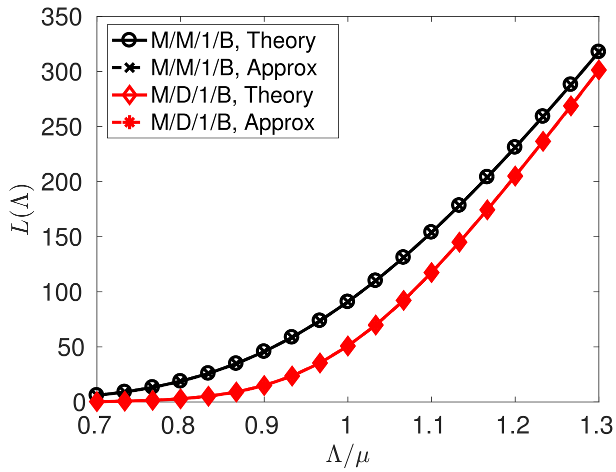


Fig. 10. Polynomial approximation of loss rate functions.

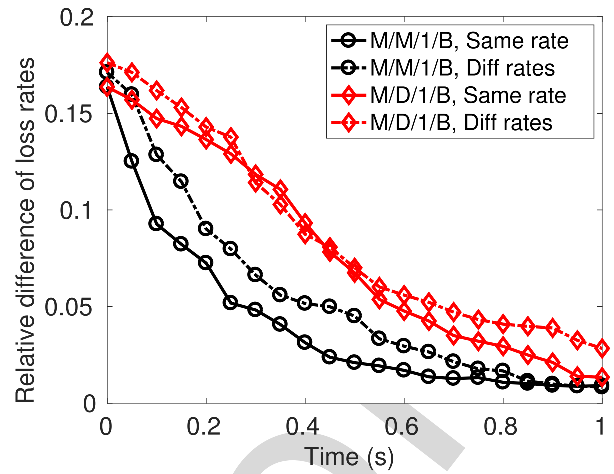


Fig. 11. Convergence performance of relative loss rates.

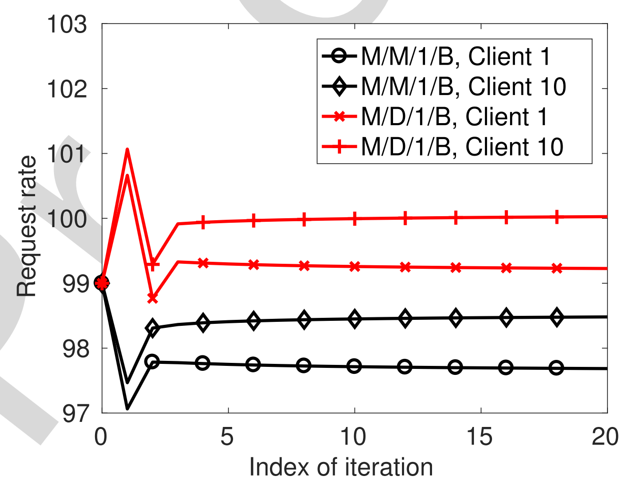


Fig. 12. Convergence performance of request rates for loss rate allocation.

Similar to delay allocation, we simulate two systems each with $N = 10$ clients and one server for loss rate allocation. The two systems correspond to an M/M/1/B queue and an M/D/1/B queue respectively. That is to say, the request arrival processes are both Poisson, while the service time distributions are exponential and deterministic respectively. The buffer size B is fixed to be 10 for each system. Besides, we set the average service rate $\mu = 1 \times 10^3\mathrm{s}^{-1}$, and the initial total average request rate $\Lambda = 0.99\mu = 0.99 \times 10^3\mathrm{s}^{-1}$.

1) Polynomial Approximation of Loss Rate Function: First, we evaluated the assumption that the loss rate function can be well approximated by a polynomial $L(\Lambda)$. Similar to delay allocation, we use theoretical results to obtain the loss rate function $\bar{L}(\Lambda)$. Recall that $\bar{L}(\Lambda) = \Lambda P_B(\Lambda)$. For the M/M/1/B queue, the blocking probability $P_B(\Lambda)$ is given by the following formula:

$$P_B(\Lambda) = \frac{(\frac{\Lambda}{\mu})^B}{\sum_{i=0}^{B}(\frac{\Lambda}{\mu})^i}.$$

For the M/D/1/B queue, $P_B(\Lambda)$ can be calculated by the procedure described in [20]. Therefore, we can get $P_B(\Lambda)$ and $\bar{L}(\Lambda)$ for any given Λ.

In our simulations, we fit $P_B(\Lambda)$ with ten samples where $\Lambda/\mu \in [0.7, 1.3]$ to be a 6-order polynomial. The total disutility functions in terms of loss rate $L(\Lambda)$ before and after approximation are compared in Fig. 10, labeled as "Theory" and "Approx" respectively. Similar to delay allocation, the polynomial approximation can be observed to match the theoretical functions very well. The largest relative error is only about 1.57%.

2) Dropping Policy: We implemented our DropSRLR dropping policy for loss rate allocation and validated the convergence of relative loss rates via simulations. To quantify the convergence performance, we introduce *the relative difference of loss rates*, defined as

$$\left(\max_i \frac{\bar{l}_i(t)}{l_i} - \min_i \frac{\bar{l}_i(t)}{l_i}\right) \Big/ \sum_i \frac{\bar{l}_i(t)}{l_i}.$$

Fig. 11 shows the evolution of the relative difference of loss rates for both systems for two sets of initial request rates. Similar to delay allocation, "Same rate" means all ten clients have the same request rate $\lambda = \Lambda/N = 99\mathrm{s}^{-1}$. On the other hand, for "Diff rates" in loss rate allocation we set $\lambda_i = 100\mathrm{s}^{-1}$ for $i = 1, 2, \ldots, 5$ and $98\mathrm{s}^{-1}$ for $i = 6, 7, \ldots, 10$. The initial loss rate of client i is set to be i. From the figure, we can see that the relative difference of loss rates converges to 0 quickly for both systems and both sets of initial request rates. It shows that our DropSRLR dropping policy ensures that the loss rates experienced are as allocated and the policy is thus efficient.

3) Distributed Protocol: We also validated the convergence of our distributed rate control protocol for loss rate allocation, Protocol 2, in our simulations. Similar to delay allocation, the utility function of client i is $U_i(\lambda_i) = \alpha w_i \log \lambda_i$. We set $\alpha = 50$ as the common scaling coefficient for all clients. We set the weights to be in two groups: $w_i = 1 - 5 \times 10^{-3}$ for $i = 1, 2, \ldots, 5$ and $1 + 5 \times 10^{-3}$ for $i = 6, 7, \ldots, 10$. For the step size, we let $\kappa(k) = 80/k$ for all k.

Fig. 12 shows the rate convergence performance for the two finite-buffer systems. In our setup, the rate evolution of Client 1 and Client 10 depicts the rate evolution of all the ten clients. We can see that for each system, the request

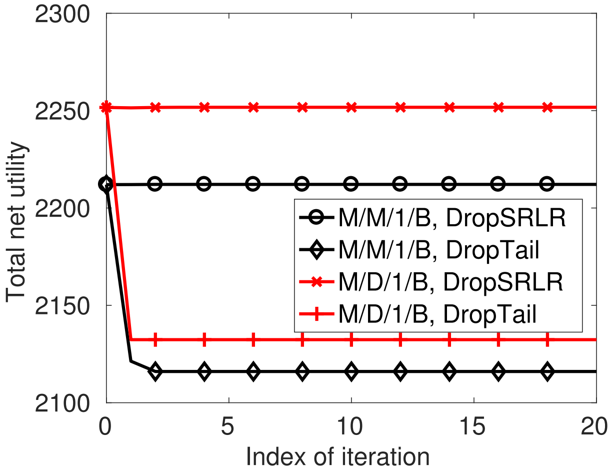


Fig. 13. Convergence performance of total net utility for loss rate allocation.

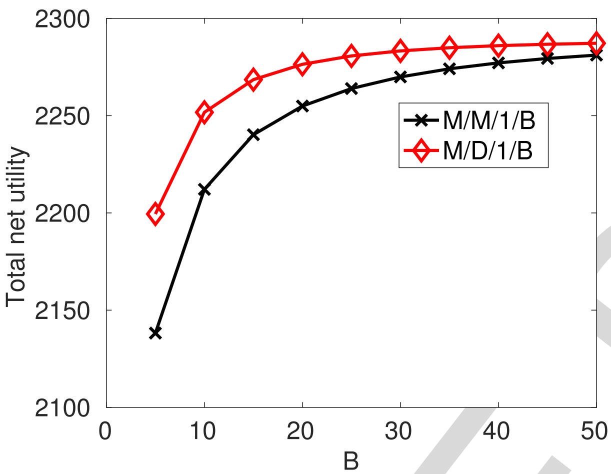


Fig. 14. Sensitivity on the buffer size B.

rates converge to two distinct values after tens of iterations. Fig. 13 shows the convergence performance in terms of total net utility for the two systems. The total net utility settles down quickly with our distributed protocol ("DropSRLR" in the figure). Note that the centralized method is omitted since it is essentially the same as the distributed protocol for loss rate allocation. Therefore, under our distributed protocol, the request rates of all clients converge to the Nash Equilibrium, and the total net utility converges to the optimal value of the rate control problem. On the other hand, under the baseline mechanism with the DropTail dropping policy the total net utility converges to a suboptimal value for each system. Hence, the loss rate allocation of the baseline mechanism is not efficient.

We also conduct sensitivity analysis on the buffer size B via simulations. The results are plotted in Fig. 14, and they clearly show diminishing returns. The total net utility, i.e. the objective value of the rate control problem, increases as the buffer size B increases. This is consistent with the intuition that larger buffer size leads to smaller loss rates. However, the marginal increase in total net utility decreases and the total net utility becomes saturated when B is large.

IX. Conclusions

We have presented our non-monetary mechanism for optimal rate control through efficient cost allocation. First, we focus on delay allocation. We give our delay allocation rule and prove its efficiency based on multinomial expansion. Then we propose our MRQ scheduling policy that can enforce the delay allocation rule effectively in the heavy traffic regime. Besides, we design a distributed rate control protocol which can lead the system to the Nash Equilibrium. Furthermore, we show that our non-monetary mechanism can be extended to handle loss rate allocation as well. Finally, simulation results depict the effectiveness of our mechanism. We will conduct further study on VIP clients for future work. We would like to obtain nontrivial sufficient conditions for clients to become VIPs and for our mechanism to still achieve efficient cost allocation considering VIP clients.

Appendix A
Proof of Lemma 1

Proof: The proof of Eq. (18) is omitted since it is virtually the same as the proof of [18, Lemma 7].

The proof of Eq. (19) is stated below:

$$\begin{aligned}
|\Delta V_{\perp}(t)| &= |\|\boldsymbol{Q}_{\perp}(t+\tau)\| - \|\boldsymbol{Q}_{\perp}(t)\|| \\
&\leq \|\boldsymbol{Q}_{\perp}(t+\tau) - \boldsymbol{Q}_{\perp}(t)\| \\
&= \left\|\boldsymbol{Q}(t+\tau) - \boldsymbol{Q}(t) - \boldsymbol{Q}_{\parallel}(t+\tau) + \boldsymbol{Q}_{\parallel}(t)\right\| \\
&\leq \|\boldsymbol{Q}(t+\tau) - \boldsymbol{Q}(t)\| + \left\|\boldsymbol{Q}_{\parallel}(t+\tau) - \boldsymbol{Q}_{\parallel}(t)\right\|.
\end{aligned}$$

The vector in the second term is exactly the projection of $\boldsymbol{Q}(t+\tau) - \boldsymbol{Q}(t)$ onto $\boldsymbol{g}$. Due to Pythagoras theorem, $\left\|\boldsymbol{Q}_{\parallel}(t+\tau) - \boldsymbol{Q}_{\parallel}(t)\right\| \leq \|\boldsymbol{Q}(t+\tau) - \boldsymbol{Q}(t)\|$. Hence,

$$\begin{aligned}
|\Delta V_{\perp}(t)| &\leq 2(\|\boldsymbol{Q}(t+\tau) - \boldsymbol{Q}(t)\|) \\
&= 2\sqrt{\sum_{i=1}^{N} \frac{1}{\hat{g}_i}(A_i(t) - S_i(t))^2} \\
&\leq 2\sqrt{\frac{N}{\hat{g}_{\min}}},
\end{aligned}$$

where the last inequality follows because we assume that there is at most one request arrival and one request service in each time slot. ■

Appendix B
Proof of Theorem 3

We will use a descent lemma in [19]:

Lemma 2: Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be continously differentiable, and let $\boldsymbol{x}$ and $\boldsymbol{y}$ be two vectors in $\mathbb{R}^n$. Suppose that

$$\|\nabla f(\boldsymbol{x} + t\boldsymbol{y}) - \nabla f(\boldsymbol{x})\| \leq Lt\,\|\boldsymbol{y}\|, \quad \forall t \in [0,1],$$

where L is some scalar. Then

$$f(\boldsymbol{x}+\boldsymbol{y}) \leq f(\boldsymbol{x}) + \boldsymbol{y}^T \nabla f(\boldsymbol{x}) + \frac{L}{2}\|\boldsymbol{y}\|^2.$$

Proof: See [19, Proposition A.24]. ■

Proof of Theorem 3: First, note the distributed protocol is possible to stop at some iteration k, if $\boldsymbol{\lambda}(k) = \boldsymbol{\lambda}^*$.

Since $\nabla f(\boldsymbol{\lambda}^*) = \mathbf{0}$, $\boldsymbol{\lambda}^*$ is stationary between successive iterations. In such case, the optimal point is reached in finite iterations. Below we will focus on the case where we have an infinite sequence $\{\boldsymbol{\lambda}(k)\}$.

Let $f(\boldsymbol{\lambda}) := \Lambda C(\Lambda) - \sum_i U_i(\lambda_i)$ be the opposite to the objective function of the server's optimization problem in (1). Easy to check f is smooth, strictly convex, and bounded on $\mathcal{S}_{\boldsymbol{\lambda}}$. Therefore, ∇f is Lipschitz-continous, i.e. there exists $L < \infty$ such that $\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \le L\|\boldsymbol{x} - \boldsymbol{y}\|, \forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}_{\boldsymbol{\lambda}}$.

By Lemma 2, we have

$$f(\boldsymbol{\lambda}(k+1)) - f(\boldsymbol{\lambda}(k)) \le \nabla^T f(\boldsymbol{\lambda}(k))(\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)) + \frac{L}{2}\|\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)\|^2 \quad (29)$$

We can rewrite the iterative update in the distributed protocol in vector form:

$$\hat{\boldsymbol{\lambda}}(k+1) = \boldsymbol{\lambda}(k) - \kappa(k)\nabla f(\boldsymbol{\lambda}(k)), \quad (30)$$

$$\boldsymbol{\lambda}(k+1) = \mathrm{P}^k(\hat{\boldsymbol{\lambda}}(k+1)), \quad (31)$$

where P^k is the projection to the convex set $\mathcal{S}^k_{\boldsymbol{\lambda}} := \{\boldsymbol{\lambda}\lambda_\delta \le \lambda_i \le \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}, \forall i = 1, 2, \ldots, N\}$. Easy to see $\mathcal{S}^k_{\boldsymbol{\lambda}} \subset \mathcal{S}_{\boldsymbol{\lambda}}, \boldsymbol{\lambda}(k) \in \mathcal{S}^k_{\boldsymbol{\lambda}}$, and $\boldsymbol{\lambda}(k+1) \in \mathcal{S}^k_{\boldsymbol{\lambda}}$.

By the Projection Theorem (See [19, Proposition 2.1.3]),

$$\left(\hat{\boldsymbol{\lambda}}(k+1) - \boldsymbol{\lambda}(k+1)\right)(\boldsymbol{\lambda} - \boldsymbol{\lambda}(k+1)) \le 0, \quad \forall \boldsymbol{\lambda} \in \mathcal{S}^k_{\boldsymbol{\lambda}}.$$

Let $\boldsymbol{\lambda} = \boldsymbol{\lambda}(k)$, and substitute in (30). We then have

$$(\boldsymbol{\lambda}(k) - \kappa(k)\nabla f(\boldsymbol{\lambda}(k)) - \boldsymbol{\lambda}(k+1))(\boldsymbol{\lambda}(k) - \boldsymbol{\lambda}(k+1)) \le 0.$$

Hence,

$$\nabla^T f(\boldsymbol{\lambda}(k))(\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)) \le -\frac{1}{\kappa(k)}\|\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)\|^2 \quad (32)$$

Substituting (32) to (29), we get

$$f(\boldsymbol{\lambda}(k+1)) - f(\boldsymbol{\lambda}(k)) \le \left(\frac{L}{2} - \frac{1}{\kappa(k)}\right)\|\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)\|^2 \quad (33)$$

Since $\kappa(k)$ satisfies $\sum_{k=0}^{\infty}\kappa^2(k) < \infty$, there must exist some integer $K_1 > 0$ such that for all $k \ge K_1$, $\kappa(k) < \frac{2}{L}$. Therefore,

$$f(\boldsymbol{\lambda}(k+1)) \le f(\boldsymbol{\lambda}(k)), \quad \forall k \ge K_1.$$

By assumption, there is a bounded optimal value for the server's optimization problem at $\boldsymbol{\lambda}^*$. Hence, $\{f(\boldsymbol{\lambda}(k))\}$ is monotonically decreasing and lower bounded by $f(\boldsymbol{\lambda}^*)$. Therefore, $\{f(\boldsymbol{\lambda}(k))\}$ converges as $k \to \infty$. Taking the limit of (33), the left hand side goes to 0, and the right hand side is nonpositive. Therefore, $\|\boldsymbol{\lambda}(k+1) - \boldsymbol{\lambda}(k)\| \to 0$ as $k \to \infty$. Since $\{\boldsymbol{\lambda}(k)\}$ is bounded in $\mathcal{S}_{\boldsymbol{\lambda}}$, the sequence must converge to some point in $\mathcal{S}_{\boldsymbol{\lambda}}$.

Let $\bar{\boldsymbol{\lambda}} \in \mathcal{S}_{\boldsymbol{\lambda}}$ be the limit point of $\{\boldsymbol{\lambda}(k)\}$ as $k \to \infty$. We shall show $\bar{\boldsymbol{\lambda}} = \boldsymbol{\lambda}^*$ by contradiction. Suppose $\bar{\boldsymbol{\lambda}} \ne \boldsymbol{\lambda}^*$, which implies $\nabla f(\bar{\boldsymbol{\lambda}}) \ne \mathbf{0}$. Hence, $\lim_{k\to\infty}\|\nabla f(\boldsymbol{\lambda}(k))\| = \|\nabla f(\bar{\boldsymbol{\lambda}})\| > 0$. Since the sequence $\{\boldsymbol{\lambda}(k)\}$ is infinite, $\|\nabla f(\boldsymbol{\lambda}(k))\| > 0$ for all k. Therefore, there exists $\varsigma_1 > 0$ such that $\|\nabla f(\boldsymbol{\lambda}(k))\| > \varsigma_1 > 0$ for all k.

Let $\Gamma(\Lambda) := \Lambda C(\Lambda)$. $\Gamma(\Lambda)$ is strictly convex and thus $\Gamma'(\Lambda)$ is strictly increasing. Besides, since $U_i(\cdot)$ is strictly concave, $U'_i(\cdot)$ is strictly decreasing. Consider $\boldsymbol{\lambda}(k)$ and $\Lambda(k) = \sum_i \lambda_i(k)$ for large k, the following are all the possible cases:

1) $\Lambda(k) = (1-\epsilon)\mu$. We know $\Lambda(k) > \Lambda^*$, and therefore $\Gamma'(\Lambda(k)) \ge \Gamma'(\Lambda^*)$. There must be some client i such that $\lambda_i(k) > \lambda_i^*$, and thus $U'_i(\lambda_i(k)) < U'_i(\lambda_i^*)$. Hence, $U'_i(\lambda_i(k)) - \Gamma'(\Lambda(k)) < U'_i(\lambda_i^*) - \Gamma'(\Lambda^*) = 0$, and the update substep will have $\hat{\lambda}_i(k+1) < \lambda_i(k)$. Since $\lambda_i(k) > \lambda_i^* > \lambda_\delta$, the distributed projection allows λ_i to decrease. Therefore, after one iteration we have $\lambda_i(k+1) < \lambda_i(k) = \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$. For all $j \ne i$, $\lambda_j(k+1) \le \lambda_j(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$. Therefore, $\Lambda(k+1) < (1-\epsilon)\mu$.
2) $\Lambda(k) < (1-\epsilon)\mu$, and there is some i such that $\lambda_i(k) = \lambda_\delta$. Recall that under efficient delay allocation, $\frac{\partial}{\partial\lambda_i}\lambda_i D_i(\lambda_i, \lambda_{-i}) = \frac{\partial}{\partial\lambda_i}\Lambda C(\Lambda) = \Gamma'(\Lambda)$. We have

$$-\frac{\partial}{\partial\lambda_i}f(\boldsymbol{\lambda}(k)) = U'_i(\lambda_\delta) - \Gamma'(\Lambda(k)) = U'_i(\lambda_\delta) - \frac{\partial\lambda_i D_i}{\partial\lambda_i}(\lambda_\delta, \lambda_{-i}(k)) > 0,$$

where the last inequality is due to the assumption that the Nash Equilibrium is in the interior of the feasible set $\mathcal{S}_{\boldsymbol{\lambda}}$. The update substep will then have $\hat{\lambda}_i(k+1) > \lambda_i(k)$. Note that $\sum_k \kappa^2(k) < \infty$ implies $\lim_{k\to\infty}\kappa(k) = 0$. Besides, $\frac{\partial}{\partial\lambda_i}f(\boldsymbol{\lambda}(k))$ is bounded. Since $\lambda_i(k) < \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$, for sufficiently large k, $\lambda_\delta < \hat{\lambda}_i(k+1) < \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$. After one iteration, we have $\lambda_\delta < \lambda_i(k+1) < \lambda_i(k)\frac{(1-\epsilon)\mu}{\Lambda(k)}$. Hence, $\Lambda(k+1) < (1-\epsilon)\mu$ and $\lambda_i(k+1) > \lambda_\delta, \forall i$.
3) $\Lambda(k) < (1-\epsilon)\mu$ and $\lambda_i(k) > \lambda_\delta, \forall i$. In this case, $\boldsymbol{\lambda}(k)$ lies in the interior of $\mathcal{S}^k_{\boldsymbol{\lambda}}$. Note that $\lim_{k\to\infty}\kappa(k) = 0$, and $\|\nabla f(\boldsymbol{\lambda}(k))\|$ is bounded. Therefore, for sufficiently large k, $\hat{\boldsymbol{\lambda}}(k+1)$ also lies in the interior of $\mathcal{S}^k_{\boldsymbol{\lambda}}$. In this case, $\boldsymbol{\lambda}(k+1) = P^k(\hat{\boldsymbol{\lambda}}(k+1)) = \hat{\boldsymbol{\lambda}}(k+1)$. Hence, $\Lambda(k+1) < (1-\epsilon)\mu$ and $\lambda_i(k+1) > \lambda_\delta, \forall i$.

Therefore, we can conclude that there exists an integer $K_2 > 0$, such that for all $k \ge K_2$, $\Lambda(k) < (1-\epsilon)\mu$, and $\lambda_i(k) > \lambda_\delta, \forall i$. $\boldsymbol{\lambda}(k+1) = \hat{\boldsymbol{\lambda}}(k+1) = \boldsymbol{\lambda}(k) - \kappa(k)\nabla f(\boldsymbol{\lambda}(k))$. Using Lemma 2 again, we have

$$\begin{aligned} f(\boldsymbol{\lambda}(k+1)) &- f(\boldsymbol{\lambda}(k)) \\ &\le -\kappa(k)\|\nabla f(\boldsymbol{\lambda}(k))\|^2 + \frac{L}{2}\kappa^2(k)\|\nabla f(\boldsymbol{\lambda}(k))\|^2 \\ &= -\kappa(k)\left(1 - \frac{L}{2}\kappa(k)\right)\|\nabla f(\boldsymbol{\lambda}(k))\|^2 \quad (34) \end{aligned}$$

Let $K_3 := \max\{K_1, K_2\}$. For all $k \ge K_3$, $\kappa(k) < \frac{2}{L}$, and there exists some $\varsigma_2 > 0$ such that $1 - \frac{L}{2}\kappa(k) > \varsigma_2$. Recall $\|\nabla f(\boldsymbol{\lambda}(k))\| > \varsigma_1 > 0$. Substituting into (34), we have

$$f(\boldsymbol{\lambda}(k+1)) - f(\boldsymbol{\lambda}(k)) < -\varsigma_1^2\varsigma_2\kappa(k), \quad \forall k \ge K_3. \quad (35)$$

Let $\varsigma := \varsigma_1^2\varsigma_2$. Taking the telescopic sum of (35) from K_3 to some $\bar{k} > K_3$, we get

$$f(\boldsymbol{\lambda}(\bar{k})) - f(\boldsymbol{\lambda}(K_3)) < -\varsigma\sum_{k=K_3}^{\bar{k}}\kappa(k).$$

Let $\bar{k} \to \infty$. We have

$$f(\bar{\boldsymbol{\lambda}}) - f(\boldsymbol{\lambda}(K_3)) < -\varsigma \sum_{k=K_3}^{\infty} \kappa(k).$$

The left hand side is bounded, while the right hand side is $-\infty$ since $\sum_k \kappa(k) = \infty$. This results in a contradiction. Hence, it is impossible that $\bar{\boldsymbol{\lambda}} \neq \boldsymbol{\lambda}^*$. In other words, $\boldsymbol{\lambda}(k) \to \boldsymbol{\lambda}^*$ as $k \to \infty$. ■

References

[1] T. Zhao, K. Ray, and I.-H. Hou, "A non-monetary mechanism for optimal rate control through efficient delay allocation," in *Proc. 15th Int. Symp. Modeling Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt)*, Paris, France, May 2017, pp. 1–8.

[2] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021," Cisco, San Jose, CA, USA, White Paper, Mar. 2017. AQ:3

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

[4] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, Mar. 1998.

[5] K. Ray and M. Goldmanis, "Efficient cost allocation," *Manage. Sci.*, vol. 58, no. 7, pp. 1341–1356, 2012.

[6] E. Altman, T. Boulogne, R. El-Azouzi, T. Jiménez, and L. Wynter, "A survey on networking games in telecommunications," *Comput. Oper. Res.*, vol. 33, no. 2, pp. 286–311, 2006.

[7] T. Alpcan and T. Başar, "A utility-based congestion control scheme for Internet-style networks with delay," in *Proc. IEEE INFOCOM*, vol. 3. Mar. 2003, pp. 2039–2048.

[8] R. Gupta, L. Vandenberghe, and M. Gerla, "Centralized network utility maximization over aggregate flows," in *Proc. 14th Int. Symp. Modeling Optim. Mobile, Ad Hoc, Wireless Netw. (WiOpt)*, May 2016, pp. 1–8.

[9] V. Ramaswamy, D. Choudhury, and S. Shakkottai, "Which protocol? Mutual interaction of heterogeneous congestion controllers," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 457–469, Apr. 2014.

[10] T. Groves, "Incentives in teams," *Econometrica*, vol. 41, no. 4, pp. 617–631, Jul. 1973.

[11] T. Baldenius, S. Dutta, and S. Reichelstein, "Cost allocation for capital budgeting decisions," *Accounting Rev.*, vol. 82, no. 4, p. 837–867, 2007.

[12] H. Moulin and S. Shenker, "Serial cost sharing," *Econometrica*, vol. 60, no. 5, pp. 1009–1037, 1992.

[13] M. V. Rajan, "Cost allocation in multiagent settings," *Accounting Rev.*, vol. 67, no. 3, pp. 527–545, 1992.

[14] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[15] L. A. Grieco and S. Mascolo, *TCP Westwood and Easy RED to Improve Fairness in High-Speed Networks*, vol. 2334. Berlin, Germany: Springer, 2002, pp. 130–146. AQ:4

[16] L. A. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 25–38, Apr. 2004.

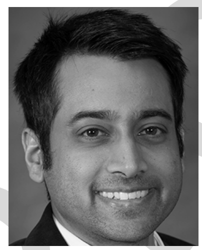
[17] D. X. Wei, C. Jin, S. H. Low, and S. Hegde, "Fast TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.

[18] A. Eryilmaz and R. Srikant, "Asymptotically tight steady-state queue length bounds implied by drift conditions," *Queueing Syst.*, vol. 72, no. 3, pp. 311–359, 2012.

[19] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA, USA: Athena Scientific, 1999.

[20] S. K. Bose. *Analysis of a M/G/1/K Queue Without Vacations*. [Online]. Available: http://home.iitk.ac.in/skb/qbook/MG1K Queue.PDF AQ:5

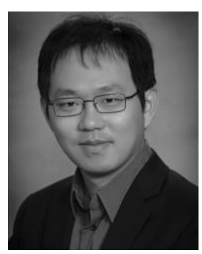
Tao Zhao received the B.Eng. and M.Sc. degrees from Tsinghua University, Beijing, China, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. His current research interests include wireless networks, cloud-based systems, and networked systems. He received the Best Student Paper Award in WiOpt 2017.



Korok Ray is currently an Associate Professor with the Mays Business School, Texas A&M University, and the Director of the Mays Innovation Research Center. He is also a Labor Economist who researches the future of work. In particular, he investigates how computer science and machine learning can create better electronic labor markets that will become ever more common in a networked society.

His core area of research is performance measurement: the study of incentives, risk/reward, and compensation for human performance. This application includes executives, chief financial officers, financial traders, farmers, doctors, teachers, rank and file employees, bankers, and even athletes. His research seeks to create economic models of human behavior and to design incentive systems to achieve better outcomes for all. His tools are economic theory, data science, and some small doses of artificial intelligence.

He received the B.S. degree in mathematics from The University of Chicago and the Ph.D. degree in economics from Stanford University. He has taught at The University of Chicago, Georgetown University, and Texas A&M University. He also served on the Council of Economic Advisers of the White House from 2007 to 2009 during the historic financial crisis.



I-Hong Hou (S'10–M'12) received the B.S. degree in electrical engineering from National Taiwan University in 2004, and the M.S. and Ph.D. degree in computer science from University of Illinois at Urbana-Champaign in 2008 and 2011, respectively.

In 2012, he joined the Department of Electrical and Computer Engineering, Texas A&M University, where he is currently an Assistant Professor. His research interests include wireless networks, wireless sensor networks, real-time systems, distributed systems, and vehicular ad hoc networks.

Dr. Hou received the Best Paper Award in ACM MobiHoc 2017, the Best Student Paper Award in WiOpt 2017, and the C.W. Gear Outstanding Graduate Student Award from the University of Illinois at Urbana-Champaign.

AUTHOR QUERIES

AUTHOR PLEASE ANSWER ALL QUERIES

PLEASE NOTE: We cannot accept new source files as corrections for your paper. If possible, please annotate the PDF proof we have sent you with your corrections and upload it via the Author Gateway. Alternatively, you may send us your corrections in list format. You may also upload revised graphics via the Author Gateway.

AQ:1 = Please confirm whether the edits made in the financial section are OK.
AQ:2 = Please note that there were discrepancies between the accepted pdf [paper.pdf] and the [paper.tex] in the sentence on lines 318–322, 854. We have followed [paper.tex].
AQ:3 = Please provide the white paper no. for ref. [2].
AQ:4 = Please confirm the volume no. for ref. [15].
AQ:5 = Please confirm the title and also provide the accessed date for ref. [20].