Impatient Queuing for Intelligent Task Offloading in Multi-Access Edge Computing

Bin Han, Vincenzo Sciancalepore, Yihua Xu, Di Feng, and Hans D. Schotten

***Abstract*—Multi-access edge computing (MEC) emerges as an essential part of the upcoming Fifth Generation (5G) and future beyond-5G mobile communication systems. It adds computational power towards the edge of cellular networks, much closer to energy-constrained user devices, and therewith allows the users to offload tasks to the edge computing nodes for low-latency applications with very-limited battery consumption. However, due to the high dynamics of user demand and server load, task congestion may occur at the edge nodes resulting in long queuing delay. Such delays can significantly degrade the quality of experience (QoE) of some latency-sensitive applications, raise the risk of service outage, and cannot be efficiently resolved by conventional queue management solutions.**

In this article, we study a latency-outage critical scenario, where users intend to limit the risk of latency outage. We propose an impatience-based queuing strategy for such users to intelligently choose between MEC offloading and local computation, allowing them to rationally renege from the task queue. The proposed approach is demonstrated by numerical simulations to be efficient for generic service model, when a perfect queue status information is available. For the practical case where the users obtain only imperfect queue status information, we design an optimal online learning strategy to enable its application in Poisson service scenarios.

***Index Terms*—Multi-access Edge Computing, task offloading, queue congestion control, impatience, online learning**

I. Introduction

While the maturity of current microelectronic technology enables portable devices to accomplish severe and complex tasks, there is a hype on the cloud computing services as enabler to offload such a burden onto ad-hoc cloud server—usually located at the edge of the network—based on the specific service requirements. This has recently fostered a novel paradigm shift towards the Edge Computing concept [1].

Edge computing (or Multi-access Edge Computing, MEC) represents the first-mover advantage for the upcoming generation of network design (5G) and the core feature of beyond-5G (B5G) networks, as it brings computation resources closer to the user equipment (UE). This enables to offload computation intensive tasks to the local edge computing node instead of the centralized mobile cloud computing (MCC) server, so that the latency can be significantly reduced [2]. With the ever-increasing demand for computing, MEC is becoming an essential solution for expected B5G use cases with very-stringent latency requirements, where a long response time of computation task may result in degraded utility or, in the worst case, a service disruption, e.g., when dealing with "age-of-information" or "age-of-task" sensitive services [3], [4].

Though widely recognized as an effective solution to resolve congestion in the central cloud, MEC itself may also be challenged by task congestion due to the complex load dynamics in realistic MEC scenarios. Various events, such as an unexpected burst of arriving computing tasks at the MEC server, several atypically time-consuming tasks, or a temporary reduction in computing capacity of the MEC server due to cross-slice resource scheduling, can overload the MEC server in a short term, and therewith result in an increased latency. Especially in the context of B5G/6G, with the prediction 5-folded increase in mobile traffic over the next decade, and the 6G ambition of all-round performance enhancement w.r.t. 5G [5], MEC congestion is not remaining a rare situation like believed so far, but becoming an emerging issue that cannot be overlooked.

There have been initial efforts made on this challenge in the recent past [6]–[9], which look for intelligent MEC offloading decision making mechanisms that mitigate MEC congestion, taking into account of the UE energy consumption and time delay. However, they generally consider only a static service scenario, where the latency is simplified to a consistent and universal (or average) value for all tasks. In practical mobile networks, to contrary, the latency is a random variable that is highly sensitive to the instantaneous server load. Furthermore, they mostly aim at optimizing the average power/latency performance over the network, while it is the outage-risk control making more critical impact in many future B5G applications, such as remote control and automation [10].

Taking another point of view, the problem of MEC server overload can also be investigated in perspective of queue congestion control. The problem of solving queue congestion has a long history in the research area of networking. A number of approaches have been proposed in the literature, such as the active queue management (AQM) technique based on queue truncation, active dropping, and preemption. AQM is proven to be the most effective approach in data networking applications such as packet routing and switching: it is capable to guarantee a full utilization of network resources, even when specified with a simple fair dropping policy [11]. Nevertheless, despite its mature development over three decades and wide success in data networking, there has been only very limited efforts such as [12] to deploy AQM in cloud computing and MEC. One significant reason is the fundamental difference between the natures of packet-level and service-level queue management: while dropping and preemption of packets in

Bin Han, Yihua Xu and Hans D. Schotten are with Technical University of Kaiserslautern, Kaiserslautern, Germany. Vincenzo Sciancalepore is with NEC Laboratories Europe, Heidelberg, Germany. Di Feng is with University of Lausanne, Lausanne, Switzerland. Bin Han (bin.han@eit.uni-kl.de) is the corresponding author.

data links can be easily remedied by automatic repeat request (ARQ) without compromising the quality of data traffic, denial of awaiting requests for cloud computing task or cancellation of an ongoing task will usually terminate the service session, and even reduce the user's future interest in the cloud service provider. This calls for a perspective change where a deep analysis of user behaviors may benefit the overall service queue management, as we have recently suggested in [13]: a detailed analysis of the well-known *impatient queuing* concept might help to prevent unexpected service disruptions while pushing for ad-hoc scheduling policies. Specifically, rational users may naturally behave impatiently if the waiting time exceeds reasonable levels (and thus failing to generate the expected end utility), and thereby hesitate to enter the queue or renege on its entrance into the queue and leave before being served [14].

To address these issues, in this paper we pioneer a risk-sensitive MEC offloading decision solution that relies on the users' impatient behavior analysis. To summarize, *i)* given an arbitrary latency distribution, we have modeled the latency-sensitive utility of a computing task, and considered the user to make risk-based offloading decision between the local computing and MEC options, *ii)* we have proposed a risk-based impatience mechanism that allows the users to flexibly regret its previous decision, and studied the conditions to exclude such regretting behavior, *iii)* we have devised optimal strategies of regretting in two typical and realistic scenarios where the aforementioned conditions are not fulfilled and, *iv)* we have validated our proposal by means of an exhaustive numerical evaluation campaign.

Compared to existing solutions, our proposal is novel regarding the following aspects:

- While most existing approaches are aiming at increasing the mean utility of tasks, we propose to make task offloading decisions with respect to the risk of latency outage, i.e. the probability of *remaining* waiting time exceeding a certain threshold.
- We consider impatient behavior of users in queues, which is rarely studied in this field but can be important for latency-critical services.
- In addition to the ideal case of Poisson (Markovian) service and perfect information of the queuing system, which is commonly investigated by literature, we consider more generic cases of i) Markov-modulated Poisson service, and ii) partial system information.

The remainder of this paper is organized as follows: we begin with a brief review to different fields related to our study in Section II, then in Section III we set up the investigated problem, and formerly discuss about the decision model of risk-based user impatience. We then analyze the user risk preference in more depth with Sections IV, proving that users will never regret in Poisson service queues, when a perfect queue status information (QSI) is available. In the same section, subsequently, we push the discussion by one step further into the cases of non-Poisson service, where a rational user may regret from its own former decision upon updated belief in the QSI. This impatient behavior is then proven in Section V to occur also in Poisson queues, when only an imperfect QSI is available to the users. For both cases of impatience, we derive the optimal regretting strategies correspondingly. The proposed approaches are then numerically evaluated in Section VI and compared against baseline solutions, before providing concluding remarks and some outlook in Section VIII.

II. Related Work

Research on the client impatience in queuing systems dates back to the 1950s, when the client behavior of hesitating to join long queues was first noticed [15], and afterwards rigorously investigated [16]. The most important pioneering work in this field was conducted by *Haight*, where the aforementioned phenomenon was named as "balking" [17], and later generalized to the "reneging" concept [18]. In 1963, *Ancker* and *Gafarian* pushed the work of *Haight* further, proposing several most widely used statistical models of balking and reneging, and therewith analyzing their impacts on $M/M/1$ queues [19], [20]. A generalization of these analyses was presented in [14] regarding $GI/G/1$ queues.

Recently, in the context of network slicing, which is an emerging use case of public cloud computing in the Fifth Generation (5G) wireless networks, we have revisited the topic of client impatience from the micro-economic perspective in a recent work [13]. We demonstrated how the statistic models of balking and reneging in $M/M/1$ queues are determined by the distribution of reward generated in the cloud service, and how this behavior will impact the resource utilization of multiple heterogeneous queues sharing a same server. Additionally, we also discussed the impact of clients' knowledge level in their decisions of balking and reneging.

Despite the commonness of assuming Poisson arrivals and services in literature of cloud computing, the validity of such assumption can be questionable in real service scenarios. Regarding complex real-world dynamics where the Poisson model fail to fit, Poisson mixture. models have been intensively studied since the 1990's. Readers with specific interest in Poisson mixture models are referred to [21]–[23].

The problem of optimal Poisson learning, which plays an important role in risk-based reneging when only imperfect status information of the Poisson queue is available (which will be studied in Section V), is strongly related to the research branch of decision making on the classic problem of "when to stop" in a learning process. Investigations to the optimal learning duration problem were initiated in the 1940s [24], [25], extended at the beginning of this century[26], and recently revisited [27], [28].

In [29], the authors introduce a model for offloading tasks with service caching by assuming that UEs can deploy the corresponding edge services onto the edge cloud before offloading their tasks, which are supposed to have some dependency among each other. Moreover, they consider the option of relaying the task offloading request to a further edge server if the closest one cannot meet the requested computational capacity requirements. The authors of [30] make similar assumptions by studying a network of edge servers and

allowing task offloading request relay, but they differentiate by accounting for delay requirements while disregarding any possible dependencies among tasks. Within the scope of task offloading, repetitive tasks stand out driven by the ubiquity of wireless sensors. Specifically, sensors may generate computationally intensive tasks in a periodic fashion, i.e. according to their duty cycle. [31] develops a game theoretical model and derives a polynomial time decentralized algorithm whereby UEs autonomously decide whether to offload their task or not in the current periodic slot. In [32], the authors disclose an algorithm to perform load balancing for IoT applications in the edge computing server with the aim of reducing the overall application response time. In particular, they consider the communication cost, computation time and average load on each edge computing server and develop two evolutionary algorithms, namely ant colony optimization and particle swarm optimization, to find a sub-optimal solution of the problem, which is proved to be NP-hard. The model assumes that each IoT node generates data following a Poisson distribution and employs traffic theory concepts to derive time-averaged performance figures, thereby considering full knowledge of static tasks and disregarding the task admission phase. Moreover, the authors do not envision for the use of machine learning to grant or reject task-offloading requests.

III. Mechanism Design

A. Task Offloading and Queuing Model

In this article we consider a generic MEC task offloading scenario, where numerous users independently and randomly generate computing tasks. When a user generates a new task, it can choose between two options: either to carry it out locally on the user device, or to offload it onto the MEC server. When locally executing the task, it takes the user a stochastic latency to obtain the result, and we consider the user to have perfect knowledge about this stochastic latency. When relying on the MEC, first it always takes a certain minimal latency $_0$ as overhead to accomplish the data transmission and signaling between the user and the MEC server. In addition to the that, all tasks offloaded by different users wait in a single task queue, and it takes every task an extra random waiting time in the service queue, until it is processed by the server. Similarly, we consider a perfect knowledge about this random queuing delay to be possessed by the MEC server. Such knowledge, either for local processing delay or MEC queuing delay, must be acquired through an accurate estimation of task execution time, which has been well studied as a key enabling technology of cloud computing regarding load balancing [33], [34] and workflow scheduling [35]. Various approaches have been proposed to address this issue on different levels [36], [37].

Without loss of generality, we consider the pending tasks to be processed by the server according to the "First Come, First Served" (FCFS) policy. The analyses and proposals in this article can be conveniently adapted to alternative queuing policies, such like "Last Come, First Served" (LCFS), random selection for service (RSS), and priority-based (PR), by replacing the stochastic model of waiting time, which is omitted in this work. For a generic mechanism design, we consider the arrivals of offloaded tasks and the service of pending tasks as independent random processes, without assuming specific distribution of inter-arrival interval or inter-service interval. We also consider the task queue to be non-preemptive with an infinite capacity. The queue offloaded tasks are usually considered in literature to be served by $c > 1$ independent servers. For the convenience of analysis, in this work we simplify the case into a single server ($c = 1$). Thus, in summary, we are studying a $G/G/1/1$ FCFS non-preemptive queue, which can be easily extended into a more general $G/G/c/1$ case where $c > 1$.

Specially, we consider the users to be possibly *impatient* in queuing. More specifically, having chosen the MEC option and waiting in the service queue, a user can regret from its previous decision at any time, retract its task from the queue without being served, and execute it locally instead. This can be identified as the well-known *reneging* behavior in queuing theory [18]. Similarly, the decision to locally carry out the task can also be understood as the *balking* behavior where the user refuses to join the queue [17].

B. Latency-Sensitive Utility Model

Every task, after being successfully processed, will generate a utility (reward) for the user that issues it. Without loss of generality, here we consider a latency-sensitive model, in which the end utility u monotonically decays along with the service latency t, asymptotically approaching 0 while staying positive, i.e:

$$u(t) > 0; \quad @u/@t < 0; \quad \forall t > 0; \tag{1}$$

$$\lim_{t \to +\infty} u(t) = 0 \tag{2}$$

where t is the *total* delay from the generation to the completion of the task, and $u(0) = u_0 > 0$ is the basic utility without latency discount.

We assume every user knows the u_0 of its own task a priori, so that at any given time t, the cumulative density function (CDF) of t, namely F_t, depends on the offloading decision z_t:

$$F_t(x|t) = \begin{cases} F_l(x|t) & z_t = 0; \\ F_c(x|t) & z_t = 1; \end{cases} \tag{3}$$

where $z_t = 0$ and $z_t = 1$ denote local computing and MEC offloading, respectively. The instantaneous CDF $F_l(x|t)$ of *remaining* local computing delay $_l$ at t is known by the user, while the instantaneous CDF $F_c(x|t)$ of *remaining* MEC computing delay $_c$ is known by the MEC server. Generally, there shall be a mechanism for the user to obtain/update the knowledge about $F_c(x|t)$, either perfect or imperfect, from the MEC server.

C. Risk-Based Impatient Queuing at MEC Server

Interestingly, in this work we investigate a risk-critical scenario, where the maximal achievable utility w.r.t. a certain

fixed outage risk P_o shall be maximized by (re-)making the offloading decision:

$$z_t = \begin{cases} 0 & \max f u(x) : 1 \quad F_l(xjt) \ 6 \ P_o g \\ & > \max f u(x) : 1 \quad F_c(xjt) \ 6 \ P_o g \\ 1 & \max f u(x) : 1 \quad F_l(xjt) \ 6 \ P_o g \\ & < \max f u(x) : 1 \quad F_c(xjt) \ 6 \ P_o g \end{cases} = \begin{cases} 0 & \min f x : F_l(xjt) > 1 \quad P_o g \\ & 6 \ \min f x : F_c(xjt) > 1 \quad P_o g \\ 1 & \min f x : F_l(xjt) > 1 \quad P_o g \\ & > \min f x : F_c(xjt) > 1 \quad P_o g \end{cases} = \begin{cases} 0 & F_l^{-1}(1 \quad P_o jt) \ 6 \ F_c^{-1}(1 \quad P_o jt) \\ 1 & F_l^{-1}(1 \quad P_o jt) > F_c^{-1}(1 \quad P_o jt) \end{cases}; \tag{4}$$

where $F_l^{-1}(xjt)$ and $F_c^{-1}(xjt)$ are the t-instantaneous inverse CDFs of remaining local and edge computing latency, respectively. As the knowledge of $F_l(xjt)$ and $F_c(xjt)$ vary as functions of t, a user may repeatedly flip its choice between local and edge computing. In practice, the latency of MEC is usually more dynamic than the local computing one, because the MEC latency is not only determined by the duration of computing the task itself, but also by the queuing latency and the network delay. Therefore, here we investigate an arbitrary task with a consistent $F_l(xjt) \overset{8t}{=} F_l(x)$ but inconsistent $F_c(xjt)$. Without loss of generality, in the remainder part of this article, when discussing a single computing task request, we always consider it to be issued at $t = 0$, if not explicitly stated otherwise.

Furthermore, to transfer the MEC server's statistical knowledge of queuing status $F_c(xjt)$ to every awaiting user in queue, there must be an additional communication mechanism. In the context of MEC, to avoid waste of radio resources, it is neither practical to hold an active RRC connection for every waiting user just to update such knowledge, nor efficient to set up a new RRC connection upon every update. One reasonable solution is to exploit the paging channel for sending the QSI to the users, which does not take any extra user plane bandwidth but a paging cost. This paging cost can be restricted to an affordable level when the QSI is only sent to a small group of users, e.g. the ones awaiting in the MEC server's task queue. However, if we want to enable a symmetric reneging, i.e. a user can also regret from its decision of local computing at any time and choose to offload its task to the MEC again, our proposed mechanism will essentially demand that every user, no matter if it is waiting in the queue or processing the task locally, keeps observing the queue status at MEC server. In this case, the QSI $F_c(xjt)$ must be broadcast to *all* users under coverage of the MEC server, which leads to a significantly increased paging cost and reduce the resource efficiency. Due to this reason, in this work we consider an asymmetric regretting procedure: a user is able to retrieve its offloaded task from the MEC queue at any time, but never regrets from a decision of local computing. In other words:

$$z_t = 0 \ ) \ z_{t'} = 0; 8t' > t: \tag{5}$$

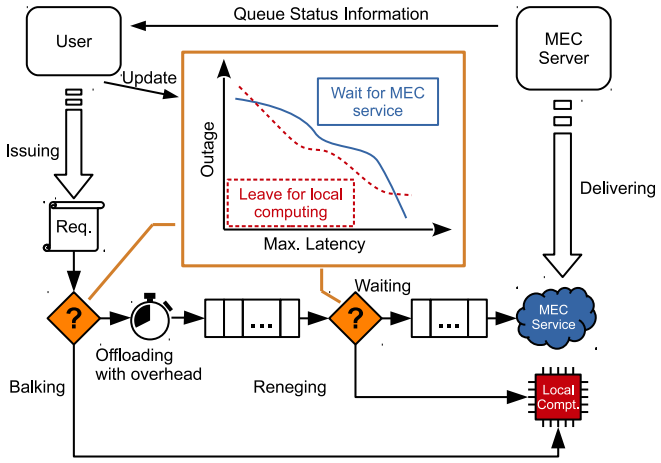


Fig. 1: An overview of the proposed risk-based user impatience mechanism

The generalization to a more generic case, where $F_l(xjt)$ is inconsistent and the user can preempt a locally undergoing task and offload it to the MEC server, will be straightforward.

Especially, under this setup, when $z_0 = 0$, the task never joins the waiting queue at MEC server, which is the balking phenomenon; when $z_0 = 1$ but $z_t = 0$ for some $t > 0$, the task joins the queue first and then leaves before being served on MEC server, which is the reneging phenomenon. This mechanism can be briefly illustrated by Fig. 1.

IV. Reneging Behavior with Perfect QSI

A. Excluding Reneging from Poisson Queues with Perfect QSI

We start our investigation with a trivial case, considering the decision of one arbitrary user under three assumptions:

1) the investigated user holds a perfect real-time information of the task queue at MEC server, including its own position in queue and the service rate of MEC server;
2) tasks are processed by the MEC server as a consistent Poisson process with rate ;
3) all users other than the investigated one are patient, or their reneging chances are significantly lower than the service rate, so that the impact of their reneging can be neglected.

When the user generates at $t = 0$ a task (but has not made the offloading decision yet), and there are already k 1 tasks in the task queue of MEC server, the remaining MEC service latency $_c$ in case of entering the queue can be modeled by $_{e;k} = _s + _{w;k}$, where $_s$ is the overhead latency to offload the task, and $_{w;k}$ is the waiting time at position k in a $G/M/1/1$ queue, of which the CDF is independent of t:

$$F_{w;k}(x) = 1 \quad \sum_{i=0}^{k-1} \frac{(\ x)^i e^{\ x}}{i!}: \tag{6}$$

Thus, $F_c(xj0)$ becomes the CDF of $_{e;k}$:

$$F_c(xj0) = F_{e;k}(x) = 1 \quad \sum_{i=0}^{k-1} \frac{[\ (x \quad _s)]^i e^{\ (x \quad _s)}}{i!}: \tag{7}$$

On the other hand, for a task that is already offloaded to the MEC server and waiting in the k^{th} position in queue at any $t > 0$, its remaining latency is simply $_{w;k}$, and $F_c(x|0)$ becomes:

$$F_c(x|t) = F_{w;k}(x) = 1 - \sum_{i=0}^{k-1} \frac{(x)^i e^{-x}}{i!}; \forall t > 0. \quad (8)$$

Clearly, with $_s > 0$ there is always $F_{e;k}(x) \leq F_{w;k}(x)$, or $F^{-1}_{e;k}(x) > F^{-1}_{w;k}(x)$, which takes the equality only when $_s = 0$, so that for all $t > 0; k \in \mathbb{N}^+; P_o \in (0;1)$:

$$F_l^{-1}(1-P_o) > F^{-1}_{e;k}(1-P_o) \Rightarrow F_l^{-1}(1-P_o) > F^{-1}_{w;k}(1-P_o); \quad (9)$$

which implies under the offloading decision rule (4) that

Lemma 1. *If a user with perfect information about a G/M/1/1 FCFS queue does not balk but enters the queue, it does not renege from the queue at its entrance position.*

Meanwhile, since $F_c(x|t) = F_{w;k}(x)$ is independent from t, obviously the reneging decision at a certain queue position k does not vary with time, i.e.

$$\begin{aligned} &\exists t_1 \in \mathbb{R}^+ : F_l^{-1}(1-P_o) > F_c^{-1}(1-P_o|t_1) \\ &\Rightarrow F_l^{-1}(1-P_o) > F_c^{-1}(1-P_o|t); \forall t \in \mathbb{R}^+. \end{aligned} \quad (10)$$

Furthermore, $F^{-1}_{w;k}(x)$ is strictly monotonically increasing w.r.t. $k \in \mathbb{N}^+$ for all $x \in (0;1)$, i.e. $F_l^{-1}(1-P_o) > F^{-1}_{w;k_1}(1-P_o) \overset{\forall k_1 > k_2}{\Rightarrow} F_l^{-1}(1-P_o) > F^{-1}_{w;k_2}(1-P_o)$. Since the position k of any specific task in a FCFS queue monotonically decreases along with time t in wide sense, i.e. $k(t_1) > k(t_2)$ for all $t_1 < t_2$. Thus, it always holds

$$\begin{aligned} &F_l^{-1}(1-P_o) > F_c^{-1}(1-P_o|t_1) \\ &\Rightarrow F_l^{-1}(1-P_o) > F_c^{-1}(1-P_o|t_2); \forall t_1 < t_2; \end{aligned} \quad (11)$$

which implies

Lemma 2. *If a user with perfect information about a G/M/1/1 FCFS queue does not renege from the queue at time t_1, it does not renege from the queue at any time $t_2 > t_1$.*

Summarizing Lemmas 1 and 2, we can conclude that the investigated user will never renege from the queue. Especially, when all users in the queue are fed with perfect QSI, reneging will be eliminated from the system, so that the assumption 3) about all other users being patient applies to every user in the system. Thus, we can conclude that:

Theorem 1. *Users do not renege from a G/M/1/1 FCFS queue with its perfect queue status information (QSI) available to all users.*

This allows us to simplify the mechanism shown in Fig. 1: the decision between MEC offloading and local computing only needs to be made once when the task is generated. The reneging behavior is excluded from the system, which is referred to as non-regretting by some literature [38].

Taking a more close view at Theorem 1, it defines two conditions for a user with consistent belief F_l in local computing delay to be non-regretting in a single-server FCFS queue:

1) The service process shall be stationary and Poisson to guarantee a memory-less $F_{w;k}$.
2) The user shall have perfect QSI, including the Poisson service rate and its real-time position $k(t)$ in the queue.

Unfortunately, both conditions can be usually violated in realistic MEC scenarios. First, against the assumption of Poisson service, inconsistent service with time-varying rate $_t$ is widely observed in field measurements. Second, making the queue status information fully transparent to all users not only generates an extra signaling overhead, but also may raise concerns about data confidentiality. Towards a queuing mechanism design in realistic scenario, we analyze both cases in the next two sections, respectively.

B. Risk-based Reneging in Poisson Mixture Queues with Perfect QSI

First we consider the complex non-Poisson queuing process where the instantaneous service rate $_t$ randomly varies over time t. This can be generically modeled as a Poisson-mixture process, a.k.a. Markov-modulated Poisson (MMP) process, where at any time t, the completion of ongoing service at the server is a Poisson random process, of which the expectation of instantaneous service rate at time t is

$$\bar{}_t = \boldsymbol{\mu} \quad \mathbf{y}_t^{\mathsf{T}}. \quad (12)$$

Here, $\boldsymbol{\mu} = \quad ^{(1)}; \quad ^{(2)} \cdots \quad ^{(D)}$ is a D-dimensional vector containing the service rate in D different states, and $\mathbf{y}_t = \left(y_t^{(1)}; y_t^{(2)} \cdots y_t^{(D)}\right)^{\mathsf{T}}$ represents the probability mass function of the server status at time t, so that $\|\mathbf{y}_t\| = 1$ for all $t \in \mathbb{R}^+$. Given the accurate server status observed at any time t_1 that $_{t_1} = \quad ^{(d)}$, we have $\mathbf{y}_{t_1} = (\underbrace{0;0;\cdots;0}_{d-1};1;\underbrace{0;0;\cdots;0}_{D-d})$, and the continuous-time Markov behavior of $\mathbf{y}_t$ implies

$$\mathbf{y}_{t_2} = \mathbf{y}_{t_1} + \int_{t_1}^{t_2} \mathbf{Q}^{\mathsf{T}} \mathbf{y} \, \mathrm{d} \quad + \boldsymbol{m}_{t_2} \quad _{t_1}; \quad (13)$$

where $\boldsymbol{m}_t$ is a $F^{\mathbf{y}}$-martingale with $\boldsymbol{m}_0 = \mathbf{0}$, $\mathbf{Q}$ is the inter-state transition rate matrix of the system. With sufficiently large D, the MMP model can accurately approximate an arbitrary time-variant service process.

Thus, at any time instant t, neglecting the impact of user reneging, the CDF of remaining waiting time at position k becomes [39]

$$F_{w;k}(x|t) = 1 - \sum_{i=1}^{k-1} \frac{\int_t^{t+x} - \mathrm{d} \quad ^i e^{\int_t^{t+x} \mathrm{d}}}{i!}; \quad (14)$$

which is jointly determined by k and $_t$. Especially, if the server status remains consistent during a time interval $[t_1; t_2]$, i.e. $_{t_i} = \quad _{t_j}$ for all $t_i; t_j \in [t_1; t_2]$, the queue is locally G/M/1/1 during $[t_1; t_2]$, so that the users will not renege, as Theorem 1 suggests. Applying this on *all* users, the system comes to a equilibrium where *every* user is patient.

When $_t$ is updated, however, $F^{-1}_{w;k}(1-P_o|t)$ may decrease and a reneging can be triggered. Therefore, we propose to update the QSI upon transitions in the service status.

It is important to remark that upon the updated queue status, all users can be impatient and have chance to renege from the queue, which may reject the assumption of negligible impact by other reneging users on the expected remaining waiting time. More specifically, an increase in $_t$ is suggesting reduced waiting time $_{w;k}$, which will *not* trigger any reneging but only inhibit the balking of new arriving users. To the contrary, a reduction in $_t$ implies longer waiting time $_{w;k}$ for all k, and thus generally encouraging the users to renege. The increased reneging rate, however, will reduce the experienced waiting time and thus partially cancel the MEC server's encouragement of reneging. Therefore, the reneging phenomenon will not further excite itself in a closed loop, but eventually achieve a equilibrium. The dynamic process of converging to the new equilibrium depends on the significance of reduction in $_t$.

In case of a slight reduction in $_t$, only a minority of reneging users will be triggered to renege. We have demonstrated in a previous study [13] of ours, that this phenomenon will not make a significant impact to the overall queue dynamics, and the assumption of $G/M/1/1$ queue can still be taken as a good approximation. However, upon a dramatic drop of the serving rate $_t$ over the queue status transition, a majority of waiting users may be simultaneously triggered to renege from the queue, while *incorrectly* assuming that all other users were patient. In this case, the users' decisions will be significantly biased and very likely non-optimal.

One possible solution to the issue caused by dense reneging is to correct the CDF model (14) regarding the impatience of users. More specifically, when estimating the remaining waiting time of a user at position k, the stochastic reneging process of all users waiting ahead of it shall be taken into account. While an analytical solution of this model is hard to obtain, it is possible to rely on the MEC server to empirically learn from its observation to the queue status and therewith online update the CDF model. The updated CDF model, when shared with the users, will also influence the user decisions and therefore change the empirical model again. Therefore, it generally becomes an incomplete information game between the MEC server and the users, which eventually converges to an equilibrium albeit the time it takes.

V. Risk-based Reneging in Poisson Queues with Imperfect QSI

Another condition that may raise user impatience is the lack of perfect QSI. In a previous work of ours [13] we have initiated a preliminary research on this case, investigating the impatient queuing behavior regarding an expected profit maximization on different levels of QSI availability. We have learned from the study that a complete blindness of users to the QSI can easily lead to inappropriate decisions of balking/reneging, and therewith impair the system performance. It turns out to be an effective compromise, nevertheless, to feed the users with an imperfect QSI, so as to enable learning-based user decisions while protecting the confidentiality of server status from users.

A. Reneging due to Estimation Errors

The mechanism of sharing imperfect QSI to users is typically implemented by periodically informing every awaiting user about the position k of its pending task in queue. A common alternative is to inform the user only when k is updated. Sometimes, a rough estimation $_0$ can also be provided to all users as an initial belief in the service rate , but the accurate value remains unknown and needs to be estimated by the user. Generally, with such mechanisms, a user is able to estimate from its observations on k, denoted by $\hat{}(\ _t)$, where

$$_t = (!_{t_1}; !_{t_2}; \dots; !_t) \tag{15}$$

is the sequence of observation entries at t, each entry $!_{t_n} = (k(t_n); t_n)$ consists of the position and the time at observation. A *consistent* estimator $\hat{}$ is guaranteed to converge at in probability:

$$\operatorname*{plim}_{t \to +1} \hat{}(\ _t) = \ ; \tag{16}$$

where plim is the probability limit operator.

Nevertheless, within a finite time t, despite the consistency of $\hat{}$ there is always a random error between the estimation $\hat{}(\ _t)$ and the ground truth :

$$"(\ _t) = \hat{}(\ _t) \ ; \tag{17}$$

which introduces an error to the decision of reneging.

More specifically, given the outage risk level P_o and the local computing latency distribution $F_l(x)$, we recall the offloading decision rule as per Eq. (4) to see that the correctness of decision relies on the user's belief $F_c(x|t)$, which is represented in Poisson queues by $F_{e;k(t)}(x)$ and determined therefore by the estimation of . If a user overestimates the service rate with $"(\ _0) < 0$ when issuing its request at $t = 0$, it will also underestimate $F^{-1}_{c;k(0)}(1 \ P_o)$, and may therewith choose to enter the queue, although it should have chosen to balk, which we name as *over-patient*; on the other hand, it can also overestimate $F^{-1}_{c;k(0)}(1 \ P_o)$ with $"(\ _0) > 0$ and therefore choose to balk, although it should have been non-regretting, which we refer to as *under-patient*.

Such imperfect decisions will, naturally, lead to increased task latency and loss in end utility. A user can rely on nothing but a longer observation sequence $_t$ to effectively reduce the estimation error, while this learning process itself (implicitly means waiting in the queue) is causing an end-utility decay over leaning time. It becomes therefore a critical task to select an optimal time to stop learning and make the reneging decision, which is discussed in the following.

B. Optimal online Poisson learning: balance between the learning gain and the waiting cost

To optimize the stopping time of learning process, we need to construct a penalty function to indicate the profit loss caused by inaccurate estimation of . As discussed above, there are two classes of error in the decision of reneging, namely over-patient and under-patient. In the earlier case, the user underestimates its waiting time in MEC server's queue, and therefore chooses to wait, expecting for an overestimated

utility, although the correct decision will reduce the latency by reneging from queue and locally computing instead. In contrast, a user in the latter case overestimates the load of MEC server and chooses to compute locally, although a rational decision, i.e. to wait, will lead to a higher utility. Thus, the expected utility loss upon decision error can be correspondingly defined as

$$L(k;t) = \begin{cases} u(t+\mathsf{E}f_{\tau_l}) - \int_0^{F_{\tau_w}(k)} u\left(t+\frac{k}{x}\right) f_\wedge(x|\Omega_t)\,\mathrm{d}x, & \lambda > \lambda_c(k); \\ \int_{\lambda_c(k)}^{+\infty} u\left(t+\frac{k}{x}\right) f_\wedge(x|\Omega_t)\,\mathrm{d}x - u(t+\mathsf{E}f_{\tau_l}), & \text{otherwise}; \end{cases} \tag{18}$$

where $\lambda_c(k)$ is the critical arrival rate, which makes $F_{\tau_w(k)}(m) = F_{\tau_l}(m)$, and $f_\wedge(x|\Omega_t)$ is the user's conditional belief about the probability density function (PDF) of estimation $\wedge$ given observations Ω_t. As the user remains waiting in the queue, Ω_t keeps being updated, and $f_\wedge(x|\Omega_t)$ converges to $\delta(x-\lambda)$, and $L(k;t)$ also therewith converges to 0.

The stochastic features of $L(k;t)$ is determined by the specific queue status information updating mechanism. As we have named in Section V-A, for example, Ω_t can be updated periodically at every $t = nT$ where $n \in \mathbb{N}$, or updated at every change of k (e.g., when the undergoing service is accomplished at server, or when other users waiting ahead renege from the queue). According to [28], the latter mechanism performs as the optimal dynamic information acquisition for Poisson random processes. Therefore, in this work we consider the queue status information updating upon service accomplishment and user reneging.

Thus, for a certain awaiting request, when its position in queue is updated at t from $k+1$ to k, it obtains a marginal learning gain of

$$G_{\text{learn}}(k;t) = L(k+1;t^-) - L(k;t); \tag{19}$$

where $t^- = \lim_{\Delta \to 0^+}(t-\Delta)$ is the left limit of time instant before the update, so that Ω_t does not contain the observation $k(t)$. It is clear that $G_{\text{learn}}(k;t)$ monotonically decreases with t, converging to 0 as $t \to +\infty$, and increases with k.

Meanwhile, the cost to further improve the estimate $\wedge$ is the expected utility loss until the next update of k, i.e. the mean inter-service interval:

$$C_{\text{learn}}(t) = u(t) - u\left(t+\frac{1}{\lambda}\right); \tag{20}$$

which is always positive and independent of k. Thus, we set the optimal decision position at

$$\begin{aligned} k_{\text{opt}}(t) &= \sup\{k : G_{\text{learn}}(k;t) - C_{\text{learn}}(t) \geq 0\} \\ &= \sup\{k : G'_{\text{learn}}(k;t) - C'_{\text{learn}} \geq 0\}; \end{aligned} \tag{21}$$

where $G'_{\text{learn}}(k;t) = G_{\text{learn}}(k;t)/u(t)$ and $C'_{\text{learn}}(k) = C_{\text{learn}}(k;t)/u(t)$.

Note that the ground truth of λ is essential for the calculation of $L(k;t)$ and C_{learn} according to Eqs. (18) and (20), but unknown to the user. So we propose to use its estimation $\wedge$ to approximate it. More specifically, we apply the bias-corrected maximum likelihood estimator

$$\wedge(\Omega_t) = \begin{cases} +\infty, & N_t = 1; \\ \frac{N_t - 1}{\sum_{n=2}^{N_t} t_n}, & N_t = 2; \\ \frac{(N_t-1)(N_t-3)}{(N_t-2)\sum_{n=2}^{N_t} t_n}, & N_t > 3; \end{cases} \tag{22}$$

where $N_t = |\Omega_t| - |\Omega_0|$ is the number of observations at t, and $t_n = t_n - t_{n-1}$ denotes the $(n-1)^{\text{th}}$ inter-update interval. Remark that since

$$t_n \sim \text{Exp}\left(\frac{1}{\lambda}\right), \quad \forall n > 2; \tag{23}$$

we have

$$\sum_{n=2}^{N_t} t_n \sim \text{Erlang}\left(N_t - 1, \frac{1}{\lambda}\right), \quad \forall N_t > 2; \tag{24}$$

which can be exploited to calculate $f_\wedge(x|\Omega_t)$.

Our proposed risk-based dynamic reneging mechanism with online learning can be briefly described by Algorithm 1.

Algorithm 1: The risk-based dynamic user reneging algorithm

```
Initialization (queue entrance at k):
  n = 0;  Ω = Ω_0; k; P_0; F_{τ_l}; ∧ = +∞; R = False
while R = False do                                              Wait
    Wait for the next observation ω_{n+1} or service
    n ← n + 1
    if Served then
        return served
    end
    k ← k − 1
    Ω ← Ω ∪ {ω_n}
    Update F_{τ_{w;k}} and ∧
    if
      G'_learn(k;t) ≥ C'_learn(k) & F^{-1}_{τ_l}(1 − P_0) ≥ F^{-1}_{τ_{w;k}}(1 − P_0)
      then
        return reneged
    end
end
return reneged
```

VI. Numerical Evaluation

A. Simulation Setup

To verify and evaluate our proposed approaches, an exhaustive simulation campaign is conducted, in order to benchmark the risk-based reneging mechanism with classical baseline methods in various service scenarios and under different information conditions.

In the simulations, we consider an exponentially decaying utility

$$u(t) = u_0 e^{-\theta t} \tag{25}$$

where θ is the discount exponent. We consider consistent Poisson arrivals of new requests at the service queue. Every request is assigned with an exponential distribution of the local computation latency, for which the mean value is individually randomly generated for each request. For the MEC service, we ignore the small delay caused by offloading overhead,

and consider two states of the server to represent different load levels, each dedicated to a specific service rate. Three different MMP scenarios are specified, each characterized by a inter-state transfer rate matrix that represents the dynamics of transition between the two aforementioned server states. Detailed specifications are listed in Tab. I.

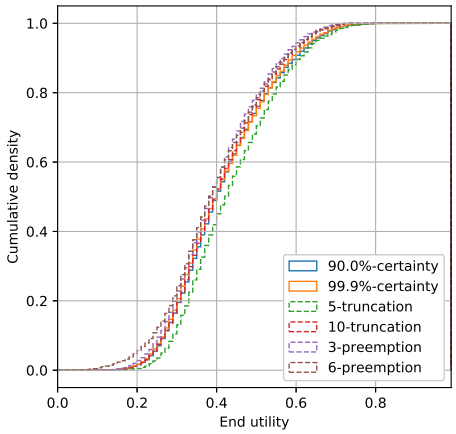
Parameter		Value	Description
		1.5	Request arrival rate
$_{l}$	Model 1: Model 2:	$U(2;10)$ $U(4;15)$	Mean local computation latency
$_{s}$		0	Task offloading overhead latency
$_{1}$		2	Service rate (low server load)
$_{2}$		1	Service rate (high server load)
Q	Scenario A: Scenario B: Scenario C:	$\begin{pmatrix}0:9 & 0:1\\ 0:8 & 0:2\end{pmatrix}$ $\begin{pmatrix}0:7 & 0:3\\ 0:3 & 0:7\end{pmatrix}$ $\begin{pmatrix}0:2 & 0:8\\ 0:1 & 0:9\end{pmatrix}$	Inter-state transition rate matrix
u_0		1	Initial utility
		0.1	Discount exponent
T_0		1	Simulation time resolution

TABLE I: Simulation setup

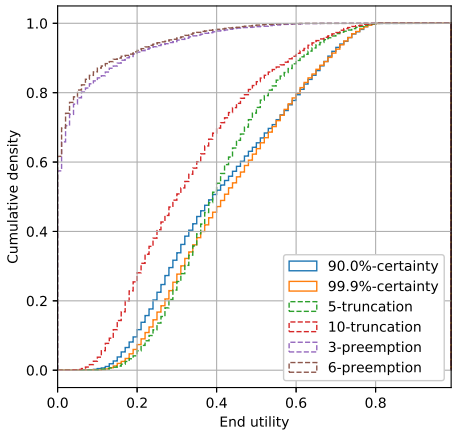
Note that according to our analysis at the beginning of Section IV-A that led us to Lemma 1, the analytical structure of the user behavior (i.e., not reneging from the queue at entrance) is valid regardless of the exact value of offloading overhead $_{s}$, as long as $_{s} > 0$. Any positive offloading overload, either random or deterministic, will shift the user belief in waiting time $_{c}$ towards higher value, while leaving the belief in $_{l}$ uninfluenced. Thus, a user is more likely to balk with a higher offloading overhead than with a lower one, if all other conditions remain the same. However, this is only a quantitative difference that makes no structural change in the user behavior. Since it is not in the scope of this work to evaluate the practical performance under realistic specification of traffic and business models, here we choose to omit the impact of $_{s}$, setting it to zero for a simple implementation and a better focus on the analytical structure.

B. Poisson Service with Perfect QSI

We started with the trivial case there the service is Poisson and a perfect QSI is available for all users, and conducted the simulation in both the service scenarios with high and low server load. In each scenario, we tested different queue congestion control approaches under variate specifications, including the risk-based reneging method with outage risk level $P_o \in \{0:1; 0:001\}$, the queue truncation method with maximal queue length $I_{\max} \in \{5; 10\}$, and the preemptive server method with preemption timeout $_{\max} \in \{3; 6\}$. More specifically, with the queue truncation method, if the instantaneous queue length $I < I_{\max}$, any arriving task will always be offloaded to the MEC server and patiently waits in the queue; when $I = I_{\max}$, any arriving task will be processed locally. With the preemption method, every task can be only processed by the MEC server for no longer than $_{\max}$, after that timeout it will be automatically removed from the server and sent back to local processing. The local computation latency model was set to model 1, i.e. $\bar{}_{l}$ $U(2;10)$. Every specific solution was evaluated through 10 independent Monte-Carlo tests, each lasting 300 periods, and the performance is measured by CDF of end utility, request admission rate, average end utility, and median end utility. The results are briefly summarized in Fig. 2 and Tab. II.



(a) Low server load, local latency model 1



(b) High server load, local latency model 1

Fig. 2: End utility distribution of Poisson queue congestion control approaches with perfect QSI

TABLE II: KPIs of Poisson queue congestion control approaches with perfect QSI

		Low server load Local latency model 1			High server load Local latency model 1		
Method		Adm	Avg	Med	Adm	Avg	Med
R	0.1%	97.57%	0.4192	0.4018	58.26%	0.4414	0.4250
	10%	97.18%	0.4222	0.4029	62.67%	0.4242	0.3957
T	5	89.34%	0.4432	0.4295	61.43%	0.4134	0.3957
	10	98,31%	0.4159	0.4029	66.60%	0.3357	0.3050
P	3	82.91%	0.4047	0.3906	74.82%	0.0525	0.0049
	6	90.72%	0.4018	0.3891	85.91%	0.0458	0.0041

Methods: risk-based Reneging; queue Truncation; server Preemption
KPIs: Admission rate; Average end utility; Median end utility

Generally, we can see that all methods work well when the server is under-loaded, i.e. when the service rate is higher than

the arrival rate. However, when the server is overloaded, i.e. when the service rate drops below the arrival rate, both the baseline methods of queue truncation and server preemption are significantly disturbed and suffer from reduced utilities. In contrast, the risk-based reneging approach outperforms the baselines with a good robustness against the congestion: it delivers not only barely influenced average/median utility, but also much heavier a tail regarding the baselines in utility distribution. In other words, compared to the baselines, the risk-based reneging method exhibits more preference to the peak-performance regime than the mid-performance regime, while generally enhancing the average performance. Furthermore, by comparing the CDF curves of 90.0%-certainty-reneging and 99.9%-certainty-reneging, we can observe that setting a higher risk level P_0 leads to a smoothed probability density of end utility (i.e. both the chances of getting high utility and low utility increase, with the chance of having intermediate utility reduced).

C. MMP Service with Perfect QSI

Then we investigated the case of non-Poisson service, where the server status randomly switches between unburdened and overloaded as a Markov process. Three such MMP scenarios A–C were defined as described in Tab. I, and in each scenario simulations were conducted with local latency specified to model 1, in the same way as in Section VI-B. Additionally, to evaluate the sensitivity to local latency, we also tested a combination of MMP scenario C and local latency model 2 where $\bar{}_1 \quad U(4, 15)$. The general results are, as summarized by Fig. 3 and Tab. III, also similar to the Poisson service case: the proposed risk-based reneging mechanism works similarly well as the baselines under moderate queue congestion (Scenario A), and gradually stands out when the congestion becomes denser (Scenario B and C). Its extra preference in peak-utility regime over mid-utility regime remains also significant here. Without any surprise, the risk-based reneging method shows a higher sensitivity to local computation latency than the baselines, tending to offload more tasks to the MEC when the local computation is slower.

D. Poisson Service with Imperfect QSI

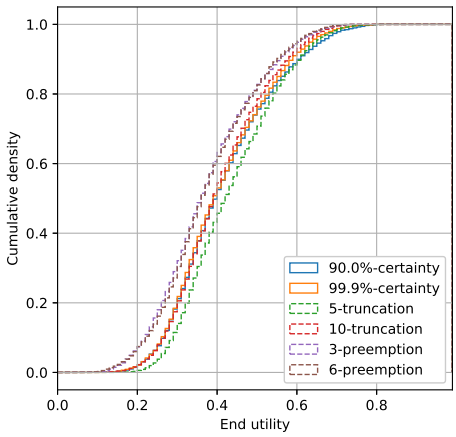
Regarding the case of Poisson service with imperfect QSI, we tested the optimal Poisson learning mechanism proposed in Section V-B. For a benchmark we also tested two static learning policies as baselines, which take at least 3 and 6 observations before considering reneging, respectively. In addition, reneging with perfect QSI is also measured as a reference for upper bound performance. Once again, the benchmark was carried out under both server load scenarios with local latency model 1, each through 10 independent 300-period Monte-Carlo tests. An additional combination of high server load and local latency model 2 was tested to evaluate the local latency sensitivity. The results are summarized in Fig. 4 and Tab. IV. It can be observed that the loss caused by improper reneging decision based on imperfect QSI is negligible when the server is under-loaded, but dramatically increases under a dense queue congestion, regardless of the learning strategy. A high sensitivity to the local latency can also be generally observed, independent from the learning strategy. Nevertheless, the optimal Poisson learning policy manages to adjust its learning phase regarding the queue status, and therewith obtains a significant utility gain over the static learning (up to 30% in average utility and 63% in median utility).

VII. Further Discussions

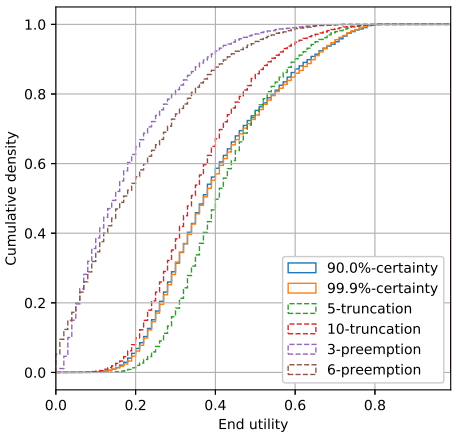
A. Inter-Task Dependency

For the convenient of model setup and analysis, in this work we have generally assumed independent arrivals and services of individual tasks. Nevertheless, inter-task dependency will not invalidate our proposed approach, as detailed below in three aspects:

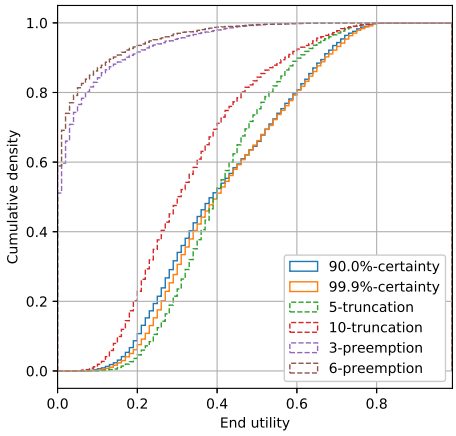
1) Inter-task arrival dependency: this will be reflected in non-Poisson arrivals of new tasks at the service queue. In FCFS queues, it is clear that the decision about impatient behavior by every user depends only on the previous tasks that had arrived before its own task, so the arrival pattern of new tasks will not make any impact on such decisions.
2) Server-state-dependent service: this will be reflected in a time-varying service rate that is determined by the server state, which can be captured by invoking the MMP model. Depending on the mechanism implementation, the instantaneous service rate may be either perfectly or only partially observable to users. The earlier case has been fully discussed in Sections IV-B and VI-C; the latter can only be resolved by an efficient online learning strategy to estimate non-Poisson processes with generically distributed queuing delay, which we identify as a direction of future study.
3) Heterogeneous services: this will be reflected in a service rate that is dependent on the service type of each individual task.
    a) In a single FCFS queue, the time-varying service rate at server is solely determined by the sequence of tasks, i.e. dependent on the random arrival sequence of heterogeneous tasks. While every user is aware of the type of its own task, it is not supposed to know the type of other tasks, and the time-varying service rate to every user can be also equivalently modeled with the MMP model. Indeed, each user is able to exploit the additional knowledge of its own task type to obtain a moderate gain by knowing the service rate it experiences at the server. This enhancement, however, is straightforward and not focused on by this work.
    b) Another possible implementation is to set up multiple queues, each only serving tasks of one specific service type, and the computing resource is shared among different queues upon the service provider's policy, as applied in our previous work [13]. In this case, the services for different queues are independent from each other under any certain



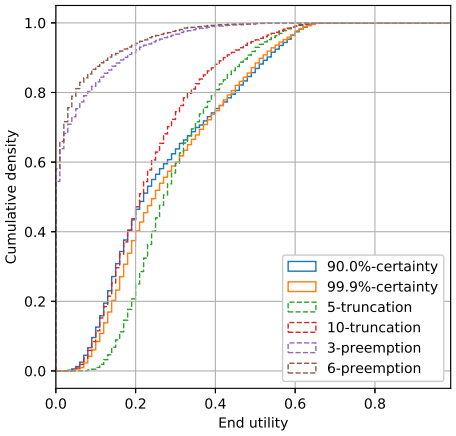
(a) Scenario A, local latency model 1



(b) Scenario B, local latency model 1



(c) Scenario C, local latency model 1



(d) Scenario C, local latency model 2

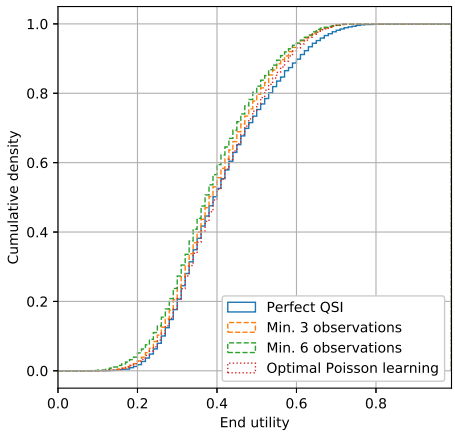
Fig. 3: End utility distribution of MMP queue congestion control approaches with perfect QSI

TABLE III: KPIs of MMP queue congestion control with perfect QSI, local computation latency specified to Model 1

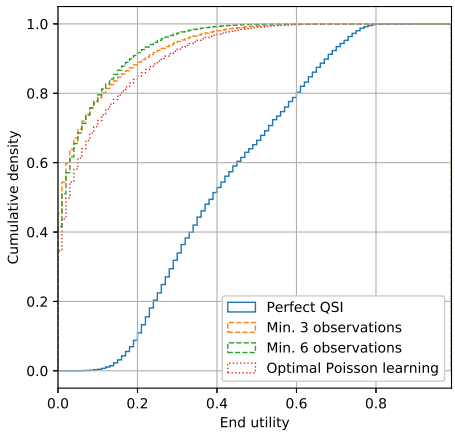
		Scenario A **Local latency model 1**			**Scenario B** **Local latency model 1**			**Scenario C** **Local latency model 1**			**Scenario C** **Local latency model 2**		
Method		Adm	Avg	Med	Adm	Avg	Med	Adm	Avg	Med	Adm	Avg	Med
R	0.1%	96.94%	0.4156	0.3971	82.66%	0.4094	0.3781	64.85%	0.4302	0.4043	70.41%	0.2905	0.2519
	10%	94.57%	0.4212	0.4008	84.33%	0.4059	0.3764	66.74%	0.4207	0.3913	71.27%	0.2782	0.2255
T	5	87.29%	0.4395	0.4252	78.41%	0.4267	0.4108	66.05%	0.4170	0.4015	65.30%	0.3015	0.2785
	10	97,43%	0.4119	0.3965	89.46%	0.3645	0.3471	72.86%	0.3395	0.3097	72.93%	0.2443	0.2176
P	3	82.76%	0.3799	0.3641	81.43%	0.1846	0.1515	76.81%	0.0546	0.0087	77.16%	0.0513	0.0070
	6	90.55%	0.3816	0.3668	89.73%	0.2076	0.1764	87.01%	0.0435	0.0055	87.01%	0.0411	0.0051

TABLE IV: KPIs of Poisson queue congestion control with imperfect QSI

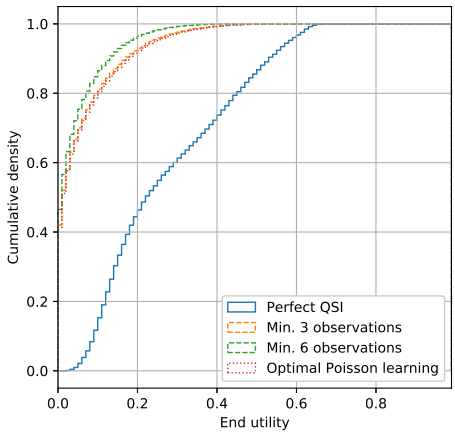
		Low server load **Local latency model 1**			**High server load** **Local latency model 1**			**High server load** **Local latency model 2**		
Method		Adm	Avg	Med	Adm	Avg	Med	Adm	Avg	Med
Optimal Poisson learning		91.91%	0.4144	0.4025	1.51%	0.0874	0.0304	1.53%	0.0594	0.0186
Fixed minimal observations	3	93.31%	0.4040	0.3894	1.24%	0.0671	0.0144	1.52%	0.0571	0.0175
	6	94.68%	0.3919	0.3772	1.52%	0.0610	0.0186	1.59%	0.0433	0.0129
Perfect QSI		96.00% (Average 1.612 obs)	0.4204	0.3992	62.23% (Average 2.530 obs)	0.4211	0.3926	64.92% (Average 2.545 obs)	0.2795	0.2275



(a) Low server load, local latency model 1



(b) High server load, local latency model 1



(c) High server load, local latency model 2

Fig. 4: End utility distribution of Poisson queue congestion control with imperfect QSI

resource allocation among queues, and each queue can be separately analyzed under the assumption of independent service among the tasks it contains.

B. Meeting Specific Latency Requirement

Another challenge that can be raised by heterogeneous services is to meet certain strict latency requirements for specific tasks that are extremely sensitive to delay, e.g. for ultra-reliable low-latency communication (URLLC) services. Our proposal in this work, indeed, does not address this issue by itself, because it does not aim at granting any certain task (nor all tasks in average) with a guaranteed quality of service over some specific lower bound. Instead, it guarantees that every individual task takes the *better* option between MEC offloading and local computing with a certainty in the decision making. If both options are promising only poor performance, e.g. when the local device has low computing power and the MEC server is also overloaded, our proposed approach cannot solely guarantee to deliver the required performance, but only help every task reduce the expected latency by taking the option that is more likely better than the other.

However, tasks generally differ from each other in *i*) the belief of local computing latency due to different local computing power, and *ii*) the belief of waiting time due to the difference position in queue. Hence, the tasks that will more likely profit from MEC offloading than local computing will tend more to stay in the queue, while the ones that are more likely to suffer from waiting latency will tend more to balk/renege. Thus, overall, the approach is capable of reducing the latency, raising the utility rate and improving the MEC efficiency in a statistical manner, as we have observed from the numerical results.

Nevertheless, in practical deployment where heterogeneous tasks in various service classes coexist and share the same MEC server, it is important to discriminate latency-tolerant tasks against latency-sensitive ones. To realize this, the aforementioned multi-queuing mechanism [13] can be introduced, where tasks are assigned to different queues regarding the service class. More specifically, the computing resource on MEC server shall be intelligently and flexibly allocated among the queues, so that the latency-sensitive queues are guaranteed with different certain expectations of waiting time regarding their priorities, and the remaining resource allocated to the latency-tolerant queue to provide an elastic service. Thus, at the the same position in queue, tasks in more prioritized queues have more left-skewed CDF of remaining waiting time than those in less prioritized queues, so that different queues are generally discriminated against each other. On top of this multi-queuing framework, our proposed risk-based reneging mechanism can be independently applied on every individual queue to further improve the intra-queue performance.

C. Multi-Objective Optimization and User Heterogeneity

It shall be remarked that in practice, heterogeneity exists not only in the service type that has been above-mentioned, but also in numerous other parameters such as device type, user mobility, energy constraint, etc. Towards a real deployment

of our proposed solution in practical MEC scenario, these parameters have to be considered by each individual user to establish its own utility function and the corresponding statistical model. Nevertheless, focusing on demonstrating the technical potentials of exploiting impatient queuing behaviors in MEC, we have omitted these details to keep the study in a generic fashion. Since the proposed decision making mechanism and framework do not rely on any specific environment or device condition, but only on the monotone of utility w.r.t. latency, their applicability will not be rejected by the practical scenario.

D. Multi-Server Scenario

While this article is focusing on a simplified case where the users decides between the options of computing the task locally and offloading it to *the only* MEC server, in practical scenario, there may be multiple MEC servers available for the users to choose. The policy to select the MEC server for initial offload of the task can be obtained by straightforwardly extending Eq. 4: the user shall take into account the believes of waiting time at *all* MEC servers as well as local computing, and take the option with lowest outage risk. However, it further introduces an additional challenge, when the users want to flexibly switch between the queues of different MEC servers regarding their instantaneous QSI. Unlike the balking/reneging behavior investigated in this article, the switching mechanism is symmetric, i.e. a user shall be allowed to go back and forth between different queues when necessary. Therefore, such a phenomenon shall be considered as another type of impatient queuing behavior: the *jockeying* [40].

Albeit the rich literature from the field of queuing theory that model and analyze the jockeying phenomenon, there has been little effort made to investigate its application and design in the context of MEC, to the best of our knowledge. Besides, existing jockeying methods, such as [41], generally aim at minimizing the expected waiting cost of users. By introducing multiple servers to the problem investigated in this article and allowing symmetric switching between servers, it can be straightforwardly extended into a possible novel design of risk-based jockeying mechanism.

VIII. Conclusion and Outlooks

In this work we have studied the risk-based user reneging mechanism, which appears a promising decentralized solution to control the MEC queue congestion with sensitivity to latency outage risk. For generic MEC tasks with time-sensitive utility, we have investigated the rational user choice between waiting within the MEC task queue and reneging for local computation, and identified the conditions of balking and reneging in Poisson service queues.

Our analysis proves that a time consistency of risk preference can sufficiently eliminate the reneging phenomenon, when a perfect statistical knowledge about queue status is available a-priori to the users. Then, we have investigated two cases where the non-regretting user behavior is violated and user reneging is activated: the non-Poisson service time and the imperfect queue status information. For non-Poisson services, we have introduced the MMP model to generalize the impatience-based user decision mechanism. Regarding the case of imperfect queue status information, we have proposed an optimal online Poisson learning strategy that balances between learning gain and waiting loss. Our proposed approaches have been verified effective in various service scenarios through exhaustive numerical simulations. The proposed novel approach allows users with computing tasks to rationally choose between MEC offloading and local computing regarding its own risk of latency outage, without having to provide its latency or utility model to the MEC service provider. It can be used as a competitive solution of decentralized queue management for multi-user scenarios that are intolerant to latency and privacy, such as autonomous driving, AI-as-a-Service, and IoT over public mobile network.

While we are focusing in this study on the utility loss caused by latency, the energy consumption of user devices also plays a key role in the practical applications of MEC task offloading. It calls for a significant follow-up effort to design a multi-objective MEC queue management framework, which supports user impatience while taking into account of the user mobility and channel condition. Besides, though we have addressed the reneging phenomenon for both scenarios of non-Poisson service and imperfect Poisson QSI, respectively, the impatience-based approach is still challenged by a more complicated case, where imperfect QSI and non-Poisson service time are combined. An efficient online learning strategy to estimate a generic distribution of queuing delay, as we have discussed in Section VII-C, is therefore required. Upon dense reneging that violates the assumption of $G/M/1/1$ queue, the incomplete information game between the MEC server and users to learn the effective queuing model online and converge to an equilibrium, which is discussed in Sec. IV-B, is worth of investigation. Additionally, in specific and practical use scenarios where heterogeneous services coexist and share the same MEC server, or when users are allowed to freely choose and flexibly switch among multiple MEC servers, it is of a great interest to evaluate the end performance of our proposed approach in combination with a multi-queuing mechanism. Furthermore, an multi-server extension that considers risk-based jockeying among different MEC servers is also a possible direction of future study.

Acknowledgment

The work of Technical University of Kaiserslautern was partly supported by the German Federal Ministry of Education and Research (BMBF) within the project Open6GHub under grant number 16KISK004. The work of NEC Laboratories Europe was supported by the European Union's Horizon 2020 project RISE-6G under grant number 101017011. The work of University of Lausanne was supported by Swiss National Science Foundation (SNSF) under grant number 100018_192583

REFERENCES

[1] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019, pp. 1468–1476.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[3] R. D. Yates, M. Tavan, Y. Hu, and D. Raychaudhuri, "Timely cloud gaming," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.

[4] X. Song, X. Qin, Y. Tao, B. Liu, and P. Zhang, "Age based task scheduling and computation offloading in mobile-edge computing systems," in *2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW)*, 2019, pp. 1–6.

[5] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The road towards 6g: A comprehensive survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021.

[6] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, 2018, pp. 1–6.

[7] X. Liu, Q. Yang, J. Luo, B. Ding, and S. Zhang, "An energy-aware offloading framework for edge-augmented mobile RFID systems," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 3994–4004, 2019.

[8] M. Huang, W. Liu, T. Wang, A. Liu, and S. Zhang, "A cloud–mec collaborative task offloading scheme with service orchestration," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5792–5805, 2020.

[9] S. Li, Q. Wang, Y. Wang, J. Xie, C. Li, D. Tan, K. Weigang, and W. Li, "Joint congestion control and resource allocation for delay-aware tasks in mobile edge computing," *Wireless Communications and Mobile Computing*, vol. 2021, pp. 1–16, 2021.

[10] B. Han, S. Yuan, Z. Jiang, Y. Zhu, and H. D. Schotten, "Robustness analysis of networked control systems with aging status," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2020, pp. 1360–1361.

[11] T. Bonald, M. Feuillet, and A. Proutiere, "Is the "law of the jungle" sustainable for the internet?" in *IEEE INFOCOM 2009*, 2009, pp. 28–36.

[12] A. H. Ismail, T. A. Soliman, G. M. Salama, N. A. El-Bahnasawy, and H. F. Hamed, "Congestion-aware and energy-efficient mec model with low latency for 5g," in *2019 7th International Japan-Africa Conference on Electronics, Communications, and Computations, (JAC-ECC)*, 2019, pp. 156–159.

[13] B. Han, V. Sciancalepore, X. Costa-Perez, D. Feng, and H. D. Schotten, "Multiservice-based network slicing orchestration with impatient tenants," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 5010–5024, July 2020.

[14] R. E. Stanford, "Reneging phenomena in single channel queues," *Mathematics of Operations Research*, vol. 4, no. 2, pp. 162–178, 1979.

[15] T. Kawata, "A problem in the theory of queues," *Rep. Statist. Appl. Res. Un. Jap. Sci. Engrs*, vol. 3, pp. 122–9, 1955.

[16] D. G. Kendall and G. H. Reuter, "The calculation of the ergodic projection for markov chains and processes with a countable infinity of states," *Acta mathematica*, vol. 97, no. 1-4, p. 103, 1957.

[17] F. A. Haight, "Queueing with balking," *Biometrika*, vol. 44, no. 3/4, pp. 360–369, 1957.

[18] ——, "Queueing with reneging," *Metrika*, vol. 2, no. 1, pp. 186–197, 1959.

[19] C. Ancker Jr and A. Gafarian, "Some queuing problems with balking and reneging–I," *Operations Research*, vol. 11, no. 1, pp. 88–100, 1963.

[20] ——, "Some queuing problems with balking and reneging—II," *Operations Research*, vol. 11, no. 6, pp. 928–937, 1963.

[21] T. Rydén, "Parameter estimation for markov modulated poisson processes," *Communications in Statistics. Stochastic Models*, vol. 10, no. 4, pp. 795–829, 1994.

[22] K. W. Church and W. A. Gale, "Poisson mixtures," *Natural Language Engineering*, vol. 1, no. 2, pp. 163–190, 1995.

[23] H. Joe and R. Zhu, "Generalized poisson distribution: the property of mixture of poisson and comparison with negative binomial distribution," *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, vol. 47, no. 2, pp. 219–229, 2005.

[24] A. Wald, "Foundations of a general theory of sequential decision functions," *Econometrica, Journal of the Econometric Society*, pp. 279–313, 1947.

[25] K. J. Arrow, D. Blackwell, and M. A. Girshick, "Bayes and minimax solutions of sequential decision problems," *Econometrica, Journal of the Econometric Society*, pp. 213–244, 1949.

[26] G. Moscarini and L. Smith, "The optimal level of experimentation," *Econometrica*, vol. 69, no. 6, pp. 1629–1644, 2001.

[27] Y.-K. Che and K. Mierendorff, "Optimal dynamic allocation of attention," *American Economic Review*, vol. 109, no. 8, pp. 2993–3029, 2019.

[28] W. Zhong, "Optimal dynamic information acquisition," *Available at SSRN 3024275*, October 2019.

[29] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2777–2792, 2021.

[30] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency iot services in multi-access edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 668–682, 2019.

[31] S. Jošilo and G. Dán, "Computation offloading scheduling for periodic tasks in mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 667–680, 2020.

[32] M. K. Hussein and M. H. Mousa, "Efficient task offloading for iot-based applications in fog computing using ant colony optimization," *IEEE Access*, vol. 8, pp. 37 191–37 201, 2020.

[33] N. K. Chien, N. H. Son, and H. D. Loc, "Load balancing algorithm based on estimating finish time of services in cloud computing," in *2016 18th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 2016, pp. 228–233.

[34] Y. Fahim, E. B. Lahmar, E. Houssine Labriji, and A. Eddaoui, "The load balancing based on the estimated finish time of tasks in cloud computing," in *2014 Second World Conference on Complex Systems (WCCS)*. IEEE, 2014, pp. 594–598.

[35] A. M. Chirkin, A. S. Belloum, S. V. Kovalchuk, M. X. Makkes, M. A. Melnik, A. A. Visheratin, and D. A. Nasonov, "Execution time estimation for workflow scheduling," *Future Generation Computer Systems*, vol. 75, pp. 376–387, 2017.

[36] P. Giusto, G. Martin, and E. Harcourt, "Reliable estimation of execution time of embedded software," in *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*. IEEE, 2001, pp. 580–588.

[37] C. Brandolese, W. Fornaciari, F. Salice, and D. Sciuto, "Source-level execution time estimation of C programs," in *Proceedings of the Ninth International Symposium on Hardware/Software Codesign*, 2001, pp. 98–103.

[38] J. Tirole, *The Theory of Corporate Finance*. Princeton University Press, 2010.

[39] S. M. Ross, J. J. Kelly, R. J. Sullivan, W. J. Perry, D. Mercer, R. M. Davis, T. D. Washburn, E. V. Sager, J. B. Boyce, and V. L. Bristow, *Stochastic Processes*. Wiley New York, 1996, vol. 2.

[40] E. Koenigsberg, "On jockeying in queues," *Management Science*, vol. 12, no. 5, pp. 412–436, 1966.

[41] A. Dehghanian, J. P. Kharoufeh, and M. Modarres, "Strategic dynamic jockeying between two parallel queues," *Probability in the Engineering and Informational Sciences*, vol. 30, no. 1, pp. 41–60, 2016.