Optimal Resource Management for Multi-access Edge Computing without using Cross-layer Communication

Ankita Koley, Chandramani Singh
Department of Electronic Systems Engineering
Indian Institute of Science
Bangalore 560012, India
Email:{ankitakoley, chandra}iisc.ac.in

***Abstract*—We consider a Multi-access Edge Computing (MEC) system with a cloud server, a base station (BS) and an MEC server attached to it. The resource constrained MEC server can be dynamically configured to serve different classes of services. The users send all the service requests to the BS which in turn keeps a subset of the requests to be served at the MEC and forwards others to the cloud server. The service requests that are processed at the MEC server incur queuing and processing delays whereas those that are sent to the cloud only incur fixed processing delays. Throughput and delay optimality warrant uplink packet scheduling at the users, MEC server configuration and service scheduling at the BS, and service forwarding to the cloud accounting for the system state. Traditional solutions to this resource management problem, e.g., those based on back-pressure , entail cross-layer message exchange. We develop two virtual queue-based drift-plus-penalty algorithms that do not require cross-layer communication, are throughput optimal, and achieve the optimal delay arbitrarily closely. The algorithms offer a tradeoff between the queuing and processing delays at the MEC server and the service processing delay at the cloud. We illustrate the performance of the algorithms via simulations.**

I. Introduction

Most of the digital applications, e.g., travel and navigation, video streaming, face recognition, augmented reality, gaming, etc., have been running using *the cloud* as the primary compute and storage resource over the last decade [1]. Cloud computing serves as a cost and energy efficient alternative to local computation, i.e., computation at users' machines. However, compute and storage demands are likely to increase manyfold in near future, e.g., Cisco has predicted that there will be 500 billion networked devices by 2030. Moreover, data centers will consume almost 15 times more energy in 2030 than their current consumption [2]. Hence, solutions solely based on cloud computing would consume extensive network resources, and will also incur huge delays, negating the benefits of cloud computing [3]. Multi-access Edge Computing (MEC) is an emerging technology in which cloud computing services are extended to edge of the Internet, i.e., wireless access points (APs) and base stations (BSs). Edge computing facilitates energy efficient and quicker execution of resource intensive tasks of users without incurring significant network resource consumption and large latencies [4].

This work was supported jointly by Centre for Network Intelligence, Indian Institute of Science (IISc), a CISCO CSR initiative and Aircel TCoE project 39010C.

MEC, specifically when it involves wireless edge networks, brings along new challenges a few of which are as follows.

(a) The edge devices, unlike the centralized cloud, have limited compute and storage capability.
(b) Many contemporary services require sizable data to be pre-stored at the edge devices.
(c) At any instant, different services have different workloads (in bits). Moreover, they also need different amount of CPU processing resources per unit of workload.
(d) Depending on the instantaneous network loads, wireless channel conditions, and resource allocation protocols, the users may receive different Quality of Service (QoS), e.g., throughput, latency etc.

Given the dynamism of various attributes, it is imperative that one judiciously decides where to offload each of the tasks, i.e., to the MEC server or to the cloud. In particular, one has to deal with the stochasticity of user requests, wireless environment and communication, compute resource allocation at the MEC server, and the delay incurred in processing the services at the cloud.

Here we develop offloading and scheduling algorithms to maximize throughput while optimizing the delay in executing the services either at the MEC server or at the cloud. In particular, we design algorithms that do not require cross-layer message exchange, stabilize the MEC system and minimize the delay.

A. Related Work

Xu et al. [5] study joint caching and task offloading at the MEC server for energy and computation delay minimization but overlook the communication resource scheduling. Poularakis et al. [6] optimize the number of services hosted by the BS while accounting for storage, computation, and communication constraints. However, they only address the static problem, representing a snapshot of the dynamic system. They primarily focus on complexity of the static problem and propose suboptimal solutions.

Dynamic user requests and network conditions warrant dynamic task offloading algorithms. Throughput optimality is an important performance criterion for dynamic task offloading algorithms. Mao et al. [7] address the problem of minimizing users' compute and communication energy subject to the

stability of user queues in an MEC system consisting of multiple users and a single MEC server. However, their model excludes the primary characteristic of MEC servers, namely, limited computation and storage resources. Cai et al. [8] consider a similar problem in a more general setup consisting of multiple users and BSs and the cloud. While Mao et al. [7] and Cai et al. [8] consider partial task offloading, i.e., allow fractions of tasks to be offloaded, their approaches entail excessive overhead. Firstly, for partial offloading the following information should be available at the users:

(a) the queue lengths at users' CPUs and at the MEC servers,
(b) the time-varying wireless channel transmission rate.

The queue length and the wireless channel state information reside at the application and the MAC/Physical layers, respectively. But communication network protocols use a layered structure which generally does not allow cross-layer communication. Although present-day mobile devices have APIs that allow the application to access wireless channel states which can be used in offloading decisions, but they still can not control the scheduling decisions that are made at the MAC layer. On the other hand, the MAC layer does not have access to the queue length information of the application layer. Secondly, there is a need to continuously exchange queue length information between the MEC servers and the users, which incurs additional resource consumption and delays. Lastly, considering the MEC server's storage constraints besides its communication constraints is essential.

Throughput-optimality has been an active research topic in the context of communication networks over the past three decades. In particular, the back-pressure algorithm which requires per flow queues and queue length information exchange across the network nodes has been proposed to stabilize multi-hop networks. Max-weight, self-regulated max-weight [9], and virtual queue length based max-weight [10] algorithms need not maintain per-flow queues and do not require queue length information exchange. But max-weight need not be throughput optimal for multi-hop networks [11], and self-regulated max-weight also stabilizes the system only for a restricted class of arrivals [9]. In contrast, the virtual queue length based max-weight is throughput optimal [10].

We consider a system with multiple classes of services or tasks, and a set of users and an MEC server associated with a BS. The users send all their tasks to the BS which either queues a task for execution at the MEC server or forwards it to the cloud. MEC comes with new traffic characteristics and QoS requirements and warrants solving joint communication and compute resource allocation problems. Since communication and computation resource allocations are activities at two different network layers, the back-pressure algorithm cannot be applied. For instance, for uplink scheduling at the MAC layer, the back-pressure algorithm requires differences between the MAC queue lengths and the per service queue lengths at the MEC server, the latter available only at the application layer.

B. Our Contribution

(a) We cast the joint stability and cost (*delay for service execution at the cloud*) minimization problem for an MEC system accounting for communication, storage, and computation constraints.
(b) We propose joint scheduling and compute and storage resource allocation algorithms that do not require per-service queue length information at the MAC layer and cross-layer queue length information exchange unlike the back-pressure based algorithms. Our algorithms use a virtual queue length-based drift plus penalty framework and offer same performance as back-pressure. .
(c) Our proofs are based on fluid limit techniques where the challenging part is to establish throughput optimality under drift plus penalty framework. Our analysis to obtain bound on the average queue length and delay for processing services at the cloud is different from the existing works.
(d) We validate our results via simulations and show the trade-off between the queuing and processing delays at the MEC server and the task execution delay at the cloud. We also compare the proposed algorithms' performance to that of a back-pressure based algorithm via simulations.

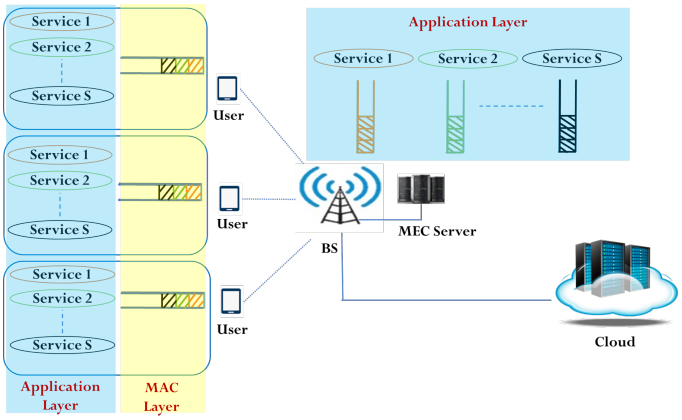


Fig. 1: An MEC System

II. System Model

We consider an MEC system with an MEC server attached to a BS and and a set of users $\mathcal{U} = \{1, 2, \cdots, U\}$ associated with the BS. The BS is also connected to the cloud through the Internet. The system evolves in discrete time, a unit of time being referred to as a *slot*.

A. Service Request Generation

The users request services from the set $\mathcal{S}=\{1, 2, \cdots, S\}$. Let $A_{s,u}(t)$ be the number of type-s services requested by user u at the beginning of slot t. We assume that all the service arrival processes at all the users are i.i.d. with $E[A_{s,u}(t)]=\lambda_{s,u}$ and $E[A^2_{s,u}(t)]<\infty$. Hence, by *the strong law of large numbers (SLLN)*, $\lim_{t\to\infty}\frac{1}{t}\sum_{\tau=1}^{t}A_{s,u}(\tau) = \lambda_{s,u}$, with probability 1. We define $A_{\max} \triangleq \max_{s,u} A_{s,u}$.

Let d_s be the number of packets generated at a user for each type-s request. Let $A_u(t)$ be the number of packets at user u at slot t and $P_u(t)$ be the cumulative count until slot t. Then $A_u(t)=\sum_{s\in\mathcal{S}}A_{s,u}(t)d_s$ and $P_u(t)=\sum_{\tau=1}^{t}A_u(\tau)$. Each user u

maintains a *first-in-first-out* (FIFO) queue, $Q_u(t)$, where the packets are stored before being scheduled for transmission to the BS.

B. Wireless Transmission to the BS

We consider a single wireless channel with J states; our analysis can easily be extended to multiple channels. We assume that the channel state process $j(t), t \geq 1$ is an irreducible, positive recurrent Markov chain with stationary distribution μ. Let $\eta_{j,u}$ be the rate achieved by user u when the channel is in state j.

C. Storage and Computation

The BS is said to have received a type-s service request once it receives all the d_s packets corresponding to that request. Let $V_{s,u}(t)$ be the number of type-s requests of user u that received by the BS at slot t. Let $A_{BS,s}(t)$ be the total number of type-s requests arrivals at slot t and $P_{BS,s}(t)$ be the cumulative count until slot t. Then $A_{BS,s}(t) = \sum_u V_{s,u}(t)$ and $P_{BS,s}(t) = \sum_{\tau=1}^{t} A_{BS,s}(\tau)$. The BS maintains a FIFO queue, $Q_{BS,s}(t)$, for each service s. It can choose to process the requests at the MEC server or to forward them to the cloud.

Let the MEC server have a CPU frequency C cycles/slot and storage R bits. If the data required for processing type-s requests is placed at the MEC server, we say that service-s is hosted by it. Let r_s denote the amount of data (in bits) required to be pre-stored at the MEC server to process type-s requests, and c_s be the CPU cycles needed to process a type-s request. We define the MEC server configuration at slot t, $X(t) = (X_s(t), s \in \mathcal{S})$, as follows.

$$X_s(t) = \begin{cases} 1, \text{ if service } s \text{ is hosted at the server at slot } t, \\ 0, \text{ otherwise.} \end{cases}$$

Clearly, $X(t) \in \mathcal{X}$ where

$$\mathcal{X} \triangleq \left\{ X \in \{0,1\}^{|\mathcal{S}|} : \sum_{s \in \mathcal{S}} r_s X_s \leq R \right\}.$$

We use $N(t) = (N_s(t), s \in \mathcal{S})$ to denote the number of services scheduled at the MEC server at slot t. We require that $\sum_s N_s(t) c_s \leq C$. So, each type-s request needs exactly one slot for processing at the MEC server. Moreover, $N(t) \in \mathcal{N}(X(t))$ where

$$\mathcal{N}(X) \triangleq \left\{ N \in \mathbb{Z}_+^{|\mathcal{S}|} : \sum_{s \in \mathcal{S}} c_s N_s \leq C,\ N_s(1 - X_s) = 0\ \forall s \right\}.$$

D. Offloading to the Cloud

The link from the BS to the cloud has capacity C_{cloud} packets/slot. We use $D_s^c(t)$ and $\tilde{D}_s^c(t)$ to denote the number of type-s requests scheduled to be forwarded to the cloud by the BS at slot t and the actual number of forwarded type-s requests, respectively. Note that $\tilde{D}_s^c(t)$ can be strictly smaller than $D_s^c(t)$ if BS does not have enough type-s requests. Also, $D_s^c(t) \in \mathcal{D}^c$ where

$$\mathcal{D}^c \triangleq \left\{ D^c \in \mathbb{Z}_+^{|\mathcal{S}|} : \sum_{s \in \mathcal{S}} d_s D_s^c \leq C_{cloud} \right\}.$$

E. Delay Cost for Service Execution at the Cloud

Let Δ_s be the fixed round trip time (RTT) incurred in sending a type-s request to the cloud, executing it there, and obtaining the result back. We define the *delay cost* for service execution at the cloud at slot t as $Y(t) = \sum_{s \in \mathcal{S}} \Delta_s \tilde{D}_s^c(t)$. We aim at minimizing

$$\limsup_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[Y(t)]$$

subject to stability of the users' and the BS's queues as defined in Section III-C.

III. Optimal Scheduling

A. Capacity Region

We first characterize the set of all service request arrival rates for which there exists an algorithm that stabilizes the system. Let $\mathcal{C}$ denote this set, also referred to as the capacity region of the MEC system. Following standard arguments (e.g., see [12]), it can be shown that

$$\mathcal{C} = \Bigg\{ \lambda \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{U}|} : \sum_{s \in \mathcal{S}} \lambda_{s,u} d_s \leq \sum_{j \in J} \mu_j \beta_{j,u} \eta_{j,u}\ \forall\, u,$$

$$\sum_{u \in \mathcal{U}} \lambda_{s,u} \leq \sum_{X \in \mathcal{X}} \pi(X) \sum_{N \in \mathcal{N}(X)} \alpha_{XN} N_s + \sum_{D^c \in \mathcal{D}^c} \phi(D^c) D_s^c\ \forall\, s,$$

$$\text{where} \sum_{u \in \mathcal{U}} \beta_{j,u} \leq 1,\ \beta_{j,u} \geq 0\ \forall j \in J\ \forall u \in \mathcal{U}$$

$$\sum_{X \in \mathcal{X}} \pi(X) \leq 1,\ \sum_{N \in \mathcal{N}(X)} \alpha_{XN} \leq 1,$$

$$\pi(X) \geq 0,\ \alpha_{XN} \geq 0\ \forall X \in \mathcal{X}, \forall N \in \mathcal{N}(X)$$

$$\sum_{D^c \in \mathcal{D}^c} \phi(D^c) \leq 1,\ \phi(D^c) \geq 0, \forall D^c \in \mathcal{D}^c \Bigg\}$$

The above characterization of the capacity region is similar to that for multi-hop networks in [10].

In the following, with a slight abuse of notation, we use $Q_u(t)$ and $Q_{BS,s}(t)$ to denote the queues as well as the respective queue lengths. We summarize the notation in Table I. Due to space constraint, we only provide outlines of the proofs which can be found in [13].

TABLE I: Notation

Symbol	Description
$\mathcal{U}$	Set of users
$\mathcal{S}$	Set of services
$\mathcal{X}$	Set of service configurations MEC server can host
$\mathcal{N}(X)$	Set of number of service configurations MEC Server can process (given the service configuration X)
$\mathcal{D}^c$	Set of service configurations can be sent to the cloud
$A_{s,u}(t)$	Number of s-type services requested by user u
$V_{s,u}(t)$	Number of s-type services requested by user u arrives at the MEC server
$Y(t)$	cost (*delay for service execution at the cloud*)
$A(t), \hat{A}(t)$	Arrival to the actual and virtual queues
$D(t), \hat{D}(t)$	Departure from the actual and virtual queues
$P(t), \hat{P}(t)$	Cumulative arrival to the actual and virtual queues
$Q(t), \hat{Q}(t)$	Actual and virtual queue lengths
Δ	Delay configuration of the services executed at the cloud
$D^c(t)$	Configuration of services to be executed at the cloud

B. Offloading and Scheduling Algorithm

Note that the system has $|\mathcal{U}|$ and $|\mathcal{S}|$ queues at the user and the server ends, respectively. For the proposed offloading and scheduling algorithms, we also maintain virtual queues corresponding to each of the $|\mathcal{U}|+|\mathcal{S}|$ queues. Let $\hat{Q}_u(t)$ and $\hat{Q}_{BS,s}(t)$ denote the lengths of the virtual queues. Further, we employ two tuneable parameters $\epsilon > 0$ and $W > 0$ that can be used to attain the desired tradeoff between the queuing and processing delays at the MEC server and the service processing delay at the cloud..

Algorithm 1 (Virtual Queue-based drift-plus-penalty)

At each user:

1) The arrival to virtual queue :
$$\hat{A}_u(t) = (1+\epsilon)\frac{P_u(t)}{t}$$
2) Scheduling policy at MAC layer:
$$u^*(t) \in \arg\max_u \hat{Q}_u(t)\eta_{j(t),u}$$
with ties broken arbitrarily
3) Departure from virtual queue at user:
$$\hat{D}_u(t) = \begin{cases} \min\{\hat{Q}_u(t), \eta_{j,u}(t)\} \text{ if } u = u^*(t) \\ 0, \text{ otherwise} \end{cases}$$
4) Departure from actual queue at user:
$$D_u(t) = \begin{cases} \min\{Q_u(t), \eta_{j,u}(t)\} \text{ if } u = u^*(t) \\ 0, \text{ otherwise} \end{cases}$$

At the MEC server:

5) Arrival to the virtual queue:
$$\hat{A}_{BS,s}(t) = (1+\epsilon)\frac{P_{BS,s}(t)}{t}$$
6) Scheduling policy at Application layer:
$$\max_{X,N,D^c} \sum_{s\in\mathcal{S}} \hat{Q}_{BS,s}(t)N_s + \sum_{s\in\mathcal{S}}(\hat{Q}_{BS,s}(t) - W\Delta_s)D_s^c$$
7) Departure from the virtual queue at the MEC server:
$$\hat{D}_{BS,s}(t) = \min\{\hat{Q}_{BS,s}(t), N_s(t) + D_s^c(t)\}$$
8) Departure from actual queue at the MEC server:
$$D_{BS,s}(t) = \min\{Q_{BS,s}(t), N_s(t) + D_s^c(t)\};$$
$$\tilde{N}_s(t) = \min\{Q_{BS,s}(t), N_s(t)\} \text{ to the MEC server}$$
and $\tilde{D}_s^c(t) = D_{BS,s}(t) - \tilde{N}_s(t)$ to the cloud

The users' actual and virtual queue lengths evolve as

$$Q_u(t+1) = Q_u(t) + A_u(t) - D_u(t) \quad (1)$$
$$\hat{Q}_u(t+1) = \hat{Q}_u(t) + \hat{A}_u(t) - \hat{D}_u(t) \quad (2)$$

Similarly, the BS's and virtual queue lengths evolve as

$$Q_{BS,s}(t+1) = Q_{BS,s}(t) + A_{BS,s}(t) - D_{BS,s}(t) \quad (3)$$
$$\hat{Q}_{BS,s}(t+1) = \hat{Q}_{BS,s}(t) + \hat{A}_{BS,s}(t) - \hat{D}_{BS,s}(t) \quad (4)$$

Clearly, $D_{BS,s}(t) \le N_s(t)+D_s^c(t)\,\forall s$. Let $\hat{P}_u(t)$ and $\hat{P}_{BS,s}(t)$ denote the cumulative arrivals to the virtual queues $\hat{Q}_u$ and $\hat{Q}_{Bs,s}$, respectively, until slot t. We assume initial condition $A_{s,u}(0)=D_u(0)=D_{BS,s}(0)=\hat{D}_u(0)=\hat{D}_{BS,s}(0)=D_s^c(0)=0$.

C. Optimality Results

Let $Q(t)=(Q_u(t), Q_{BS,s}(t))$ and $\hat{Q}(t)=(\hat{Q}_u(t), \hat{Q}_{BS,s}(t))$ denote the queue length vectors of the actual and the virtual queues at slot t. Let $P(t)=(P_u(t), P_{BS,s}(t))$ be the vector of cumulative arrivals at the actual queues. Let us also define $\mathcal{Z}(t)=\left(Q(t), \hat{Q}(t), \frac{P(t)}{t+1}\right)$. $\mathcal{Z}(t), t \ge 0$ evolves as a Markov chain under Algorithm 1. We call this Markov chain stable if it is *positive Harris recurrent* [14, Section 3]. Note that

$$\|\mathcal{Z}(t)\|_1 = \|Q(t)\|_1 + \|\hat{Q}(t)\|_1 + \frac{1}{t+1}\|P(t)\|_1.$$

We use $\mathcal{Z}^{(z)}(t), t\ge 0$ to refer to the Markov process with initial condition $\|\mathcal{Z}(0)\|_1=z$. To analyze the stability of the Markov chain $\mathcal{Z}(t)$, we consider a sequence of scaled Markov chains $\frac{1}{z}\mathcal{Z}^{(z)}(zt), z=1,2,\cdots$. The following lemma characterizes positive Harris recurrence of $\mathcal{Z}(t)$ in terms of the scaled processes.

Lemma 1. *[10, Lemma 6] Let there be $\xi, T > 0$ such that for any sequence of processes $\frac{1}{z}\mathcal{Z}^{(z)}(zt), t \ge 0, z = 1,2,\cdots$, we have $\limsup_{z\to\infty} \mathbb{E}\left[\frac{1}{z}\|\mathcal{Z}^{(z)}(zT)\|_1\right] \le 1-\xi$. Then the Markov chain $\mathcal{Z}^{(z)}(t), t \ge 0$ is stable.*

The following theorem shows that the scaled Markov chains $\frac{1}{z}\mathcal{Z}^{(z)}(zt), t \ge 0, z = 1,2,\cdots$ satisfy the condition for stability as in Lemma 1 under Algorithm 1.

Theorem 2. *The MEC System is stable under Algorithm* 1 *for any arrival rate vector λ such that $(1+\epsilon)\lambda \in \mathcal{C}$.*

Proof: (*outline*) To prove stability of the MEC system, we first prove that the virtual queues are stable. Following Algorithm 1, after large enough time the arrival rates at the virtual queues become very close to $(1+\epsilon)\lambda$. Hence, stability of the virtual queues implies that the service rates for the virtual queues are at least $(1+\epsilon)\lambda$. Since the service rates of both the virtual and the actual queues are same, stability of the virtual queues implies stability of the actual queues.

Arrivals at the virtual queues at the users and at the MEC server consitute single-hop traffic. The packets in the virtual queues leave the network after service without getting transferred to the subsequent hop queues. Hence, the virtual queues decompose the multi-hop network into two single-hop networks. Therefore any exchange of queue lengths from the MEC server to the users is not required. Following are the main technical steps of the stability proof.

The process $\{\mathcal{Z}(t), t \ge 0\}$ is scaled in both time and space by a sequence of integers $\{z_n\}$ such that $z_n \to \infty$ as $n \to \infty$ and the limit of the scaled processes $\{\frac{1}{z_n}\mathcal{Z}^{(z_n)}(z_n t), t \ge 0\}$ is referred to as the fluid limit process, $z(t)$ [10], [15].

(a) We show using SLLN that with probability 1
$$\limsup_{n\to\infty} \frac{1}{z_n}\left\lceil \frac{1}{z_n t+1}\|P^{(z_n)}(z_n t)\|_1 \right\rceil = 0. \quad (5)$$

(b) We show that with probability 1 for any $\zeta > 0$ there exists a finite time T such that
$$\limsup_{n\to\infty} \frac{1}{z_n}\sum_s \hat{Q}_u^{(z_n)}(z_n t) < \zeta\,\forall\, t \ge T. \quad (6)$$

(c) We show that with probability 1 for any $\zeta_1 > 0$ there exists a finite time T_1 such that

$$\limsup_{n\to\infty} \frac{1}{z_n} \sum_s \hat{Q}^{(z_n)}_{BS,s}(z_n t) < \zeta_1 \, \forall \, t \geq T_1. \tag{7}$$

(d) Using the above results for virtual queues we show that there exists a finite time $T^* \geq \max\{T, T_1\}$ such that with probability 1

$$\limsup_{n\to\infty} \frac{1}{z_n} \big(\sum_u Q^{(z_n)}_u(z_n t) + \sum_s Q^{(z_n)}_{BS,s}(z_n t) \big) = 0 \, \forall \, t \geq T^*. \tag{8}$$

(e) We obtain Theorem 2 by combining the above results and by uniform integrability of the scaled sequence $\{\frac{1}{x}\mathcal{Z}^{(x)}(xT^*), x = 1, 2, ..\}$.

The proofs of Steps (a) and (d) are as in [10]. Step (b) also follows from a similar technique as in [15]. Our main contribution is the proof of Step (c). For an MEC system without cloud (i.e., when the scheduling policy at the MEC server is $\max_{X,N} \sum_s \hat{Q}_{BS,s}(t) N_s$), (7) follows from arguments similar to those in [10]. But we need more delicate analysis for the MEC system with cloud. We use the observation that the scaled limit of the cost $W \sum_s D^c_s(t) \Delta_s$ is 0 since W, Δ_s and $D^c_s(t)$ are all bounded. ∎

1) Queue-Length and Cost Bounds: Let $Y_{opt}(\lambda)$ be the minimum time average expected *delay cost* that can be achieved by any control policy that stabilizes the system under the arrival rate λ. Let $(1+\epsilon)\lambda \in \mathcal{C}$. Then there exists a small enough number $\epsilon_1 > 0$ such that $(1+\epsilon)\lambda + \underline{1}\epsilon_1 \in \mathcal{C}$. Therefore there exists a stationary scheduling policy $\bar{\pi}$ such that [16]

$$\mathbb{E}[N^{\bar{\pi}}_s(t) + D^{c\,\bar{\pi}}_s(t)] \geq (1+\epsilon) \sum_u \lambda_{s,u} + \epsilon_1 \, \forall s \tag{9}$$

$$\mathbb{E}[Y^{\bar{\pi}}(t)] = Y_{opt}((1+\epsilon)\lambda + \underline{1}\epsilon_1) \tag{10}$$

where $N^{\bar{\pi}}_s(t)$ and $D^{c\,\bar{\pi}}_s(t)$ are the numbers of type-s services served by the MEC server and sent to the cloud, respectively, and $Y^{\bar{\pi}}(t)$ is the *delay cost* at slot t under policy $\bar{\pi}$.

Theorem 3. *We obtain the following bounds for the average expected delay cost and the average expected weighted virtual queue length at the MEC server.*

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[Y(t)] \leq \frac{B_2}{W} + Y_{opt}((1+\epsilon)\lambda + \underline{1}\epsilon_1)$$

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\big[\sum_s \hat{Q}_{BS,s}(t) \sum_u \lambda_{s,u} \big] \leq \frac{B_2}{\epsilon_1} + \frac{W Y_{opt}((1+\epsilon)\lambda + \underline{1}\epsilon_1)}{\epsilon_1}.$$

where $B_2 = SU(1+\epsilon)^2 A^2_{\max} d^2_{\max} + S N^2_{\max} + S D^2_{\max}$.

Proof: Let us consider a Lyapunov function $V(t) = \frac{1}{2}\sum_s \hat{Q}^2_{BS,s}(t)$ and define Lyapunov drift $\Delta V(t) = \frac{1}{2}\sum_s \hat{Q}^2_{BS,s}(t+1) - \frac{1}{2}\sum_s \hat{Q}^2_{BS,s}(t)$. Then

$$\begin{aligned}
&\Delta V(t) + WY(t) \\
=&\frac{1}{2} \sum_s \Big(\big(\hat{Q}_{BS,s}(t) - N_s(t) - D^c_s(t) \big)^+ + \hat{A}_{BS,s}(t) \Big)^2 \\
&- \frac{1}{2} \sum_s \hat{Q}^2_{BS,s}(t) + WY(t) \\
\leq&\frac{1}{2} \sum_s \Big(\hat{A}^2_{BS,s}(t) + N^2_s(t) + D^{c2}_s(t) \Big) \\
&+ \sum_s \hat{Q}_{BS,s}(t) \Big(\hat{A}_{BS,s}(t) - N_s(t) - D^c_s(t) \Big) + WY(t) \\
\leq& B_2 + \sum_s \hat{Q}_{BS,s}(t) \Big((1+\epsilon) \frac{\sum_{\tau=1}^t A_s(\tau)}{t} \Big) \\
&- \sum_s \hat{Q}_{BS,s}(t) \big(N_s(t) + D^c_s(t) \big) + WY(t)
\end{aligned} \tag{11}$$

where $A_s(t) = \sum_u A_{s,u}(t)$ and $B_2 = SU(1+\epsilon)^2 A^2_{\max} b^2_{\max} + S N^2_{\max} + S D^2_{\max}$. Let us define

$$\Theta(T) \triangleq \sum_{t=1}^{T} \mathbb{E}\left[\sum_s \hat{Q}_{BS,s}(t) \big(\frac{\sum_{\tau=1}^t A_s(\tau)}{t} - \sum_u \lambda_{s,u} \big) \right].$$

Adding and subtracting $(1+\epsilon)\sum_s \hat{Q}_{BS,s}(t) \sum_u \lambda_{s,u}$ from (11), and using (9) and (10) we obtain the following bounds.
Bound on the average expected cost:

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}[Y(t)] \leq \frac{B_2}{W} + Y_{opt}((1+\epsilon)\lambda + \underline{1}\epsilon_1) + (1+\epsilon) \limsup_{T\to\infty} \frac{1}{T} \Theta(T). \tag{12}$$

Bound on the average expected weighted queue length:

$$\limsup_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\big[\sum_s \hat{Q}_{BS,s}(t) \sum_u \lambda_{s,u} \big] \leq \frac{B_2}{\epsilon_1} + \frac{W Y_{opt}((1+\epsilon)\lambda + \underline{1}\epsilon_1)}{\epsilon_1} + (1+\epsilon) \limsup_{T\to\infty} \frac{1}{T} \Theta(T). \tag{13}$$

Both the bounds follow using standard arguments as in [16, Chapter 3]. Their proofs can be found in the extended version [13, Appendinx C]. The nonstandard part is proving that $\limsup_{T\to\infty} \frac{1}{T}\Theta(T) = 0$. Below, we provide the key arguments of this proof.

Firstly, Harris positive recurrence of the Markov chain $\mathcal{Z}(t)$ implies that $\exists$ a petite set $B = \{\mathbf{z} \in \mathcal{Z} : ||\mathbf{z}||_1 \leq \kappa\}$ for some $\kappa > 0$ and $\exists$ a $T > 0$ such that $\sup_{z\in B} \mathbb{E}_z[\tau_B(T)] < \infty$, where $\tau_B(T) = \inf\{t \geq T : \mathcal{Z}(t) \in B\}$ [14, Theorem 3.1]. This further implies that $\sum_s \hat{Q}_{BS,s}(t) \leq \kappa + \tau_B(T)(1+\epsilon) A_{\max}$ and $\mathbb{E}[\sum_s \hat{Q}_{BS,s}(t)] \leq \kappa + \mathbb{E}[\tau_B(T)](1+\epsilon) A_{\max} < \kappa_1$ for some $\kappa_1 > 0$.

Secondly, since the arrivals satisfy *SLLN, for a given* $\delta > 0$ $\exists$ *a* T_δ *such that* $\left| \frac{\sum_{\tau=1}^t A_s(\tau)}{t} - \sum_u \lambda_{s,u} \right| \leq \delta \, \forall \, t \geq T_\delta$ *with probability 1.*

Now, we divide $\Theta(T)$ *into two parts* $\sum_{t=1}^{T_\delta} \mathbb{E}\left[\sum_s \hat{Q}_{BS,s}(t)\left(\frac{\sum_{\tau=1}^{t} A_s(\tau)}{t} - \sum_u \lambda_{s,u}\right)\right]$ *and* $\sum_{t=T_\delta+1}^{T} \mathbb{E}\left[\sum_s \hat{Q}_{BS,s}(t)\left(\frac{\sum_{\tau=1}^{t} A_s(\tau)}{t} - \sum_u \lambda_{s,u}\right)\right]$. *The first term is finite since* T_δ *is finite and the second term could be made arbitrarily small by choosing a small enough* δ. *Then taking* $T \to \infty$ *we obtain* $\limsup_{T\to\infty} \frac{1}{T}\Theta(T) = 0$. ∎

By choosing $\epsilon_1 > 0$ arbitrarily small we can obtain static policies that stabilize the MEC system and yield a cost arbitrarily close to $Y_{opt}((1+\epsilon)\lambda)$. It is clear from Theorem 3 that increasing W decreases the upper bound on the average expected *delay cost* and increases the average expected virtual queue length. This could be understood as follows. Algorithm 1 chooses the schedules at the MEC server based on the optimization

$$\max_{\substack{X\in\mathcal{X}, N\in\mathcal{N}(X)\\ D^c\in\mathcal{D}^c}} \sum_{s\in\mathcal{S}} \hat{Q}_{BS,s}(t)N_s + \sum_{s\in\mathcal{S}} \left(\hat{Q}_{BS,s}(t) - W\Delta_s\right) D_s^c$$

which can be decomposed into two parts:

$$N_s(t) \in \underset{X\in\mathcal{X}, N\in\mathcal{N}(X)}{\arg\max} \sum_{s\in\mathcal{S}} \hat{Q}_{BS,s}(t)N_s, \tag{14}$$

$$D^c(t) \in \underset{D^c\in\mathcal{D}^c}{\arg\max} \sum_{s\in\mathcal{S}} \left(\hat{Q}_{BS,s}(t) - W\Delta_s\right) D_s^c. \tag{15}$$

Clearly, $D_s^c(t)$ can be positive only if $\hat{Q}_{BS,s}(t) > W\Delta_s$. For a given arrival rate, higher value of W implies less number of services being sent to the cloud. Therefore, the service rates of the virtual queues decrease. If the arrival rates are close to the capacity region boundary, a higher value of W implies higher virtual queue lengths. Since the service rates of the virtual and the actual queues are same as per Algorithm 1, the queue lengths of the actual queues also increase with W. By Little's Law the average delay is proportional to the average queue lengths. So, the parameter W facilitates a trade-off between the queuing and processing delays at the MEC server and the *d*elay cost (service execution delay at the cloud).

Remark 1. *While our fluid limit based analysis is similar to the one in Ji et al. [10], we consider a more general scenario:*

(a) *In addition to proving the stability of the queues, we also obtain an upper bound on the average* d*elay cost.*

(b) *Unlike [10] who assume a deterministic network with fixed link rates, we consider a stochastic wireless network with time varying channels.*

D. An Alternative Algorithm

Here, we propose another algorithm where the scheduler solves the same optimization problems, but the services sent to the cloud differ from Algorithm 1.

Algorithm 2 (Virtual Queue-based Drift-plus-penalty with controlled departure)*:* This Algorithm is same as Algorithm 1 except 7 and 8 are replaced by 7* and 8*.

7*) Departure from the virtual queue at the MEC server:

$$\hat{D}_{BS,s}(t) = \hat{N}_s(t) + \hat{D}_s^c(t) \text{ where}$$
$$\hat{N}_s(t) = \min\{Q_{BS,s}(t), N_s(t)\} \text{ and}$$
$$\hat{D}_s^c(t) = \min\{\left(\hat{Q}_{BS,s}(t) - \hat{N}_s(t) - W\Delta_s\right)^+, D_s^c(t)\}$$

8*) Departure from actual queue at the MEC server:

$$D_{BS,s}(t) = \tilde{D}_s^c(t) + \tilde{N}_s(t) \text{ where}$$
$$\tilde{N}_s(t) = \min\{Q_{BS,s}(t), N_s(t)\} \text{ to the MEC server}$$
$$\tilde{D}_s^c(t) = \min\{\left(Q_{BS,s}(t) - N_s(t) - W\Delta_s\right)^+, D_s^c(t)\} \text{ to the cloud}$$

Clearly, the difference between Algorithm 1 and Algorithm 2 lies only in the departures from the virtual and the actual queues at the MEC server.

Theorem 4. *The MEC Network is stable under Algorithm 2 for any arrival rate vector* λ *such that* $(1+\epsilon)\lambda \in \mathcal{C}$. *Moreover, Algorithm 2 offers the same queue-length and cost bounds as in Theorem 3 except* B_2 *being replaced with* $B_2 + S(N_{\max} + W\Delta_{\max} + D_{\max})^2$.

Proof: The proof of this theorem is along the same lines as the proofs of Theorem 2 and Theorem 3. ∎

Complexity of the scheduling algorithms: The optimization problem at the MEC server under Algorithm 1 is NP-Hard. Let us consider the first part of the problem (i.e., (14)). For each $X\in\mathcal{X}$ we need to solve the following optimization problem.

$$P1: \quad \max \sum_{s\in\mathcal{I}_X} \hat{Q}_{BS,s}(t)N_s$$
$$\text{subject to} \sum_{s\in\mathcal{I}_X} N_s c_s \le C$$
$$N_s \in [N_{\max}] \ \forall s \in \mathcal{I}_X$$

where $\mathcal{I}_X = \{s : X_s \neq 0\}$. $P1$ is a bounded Knapsack problem, a known NP-Hard problem [17]. This can be solved using Dynamic Programming (DP) [17]. Further, the set $\mathcal{X}$ can be obtained using the brute-force search method. As the number of services increases, the complexity of finding the set $\mathcal{X}$ increases exponentially. We leave the development of a low complexity algorithm as our future work. However, (14) can be approximately solved by the approach in [6]. The second part of the problem (i.e., (15)) has the same structure as using $P1$ and can also be solved by DP. Since the optimization problem at the MEC server is same for both, Algorithms 1 and 2, Algorithm 2 is also NP-Hard.

IV. Back-pressure Scheduling

To benchmark the performance of Algorithms 1 and 2, we also develop a back-pressure (BP) based scheduling algorithm which requires per-flow queue length information at each hop. As opposed to our model where each user maintains only one queue, here each user has S queues, one per service type. Let $Q_{s,u}(t)$ be the queue length for the type-s services at user u. Let $A_{s,u}$ and $D_{s,u}(t)$ be the type-s service arrivals at user u and packet departures from $Q_{s,u}(t)$, respectively, at slot t. Then $Q_{s,u}(t+1) = Q_{s,u}(t) + A_{s,u}(t)d_s - D_{s,u}(t)$. The MEC server maintains per-service queues $Q_{BS,s}(t)$ as before.

BP-based drift-plus-penalty algorithm

At each user:

1) Scheduling policy at MAC layer:
$w_u(t) = \max_s (Q_{s,u}(t) - d_s Q_{BS,s}(t))^+$

$u^*(t) \in \arg\max_u w_u(t)\eta_{j(t),u}$
$s^*(t) \in \arg\max_s (Q_{s,u^*(t)}(t) - d_s Q_{BS,s}(t))^+$

2) Departure from user:

$$D_{s,u}(t) = \begin{cases} \min\{\hat{Q}_{s,u}(t), \eta_{j,u}(t)\} \text{ if } u = u^*(t), s = s^*(t) \\ 0, \text{ otherwise} \end{cases}$$

At MEC server:

1) Scheduling policy at Application layer:

$$\max_{X,N,D^c} \sum_{s\in\mathcal{S}} Q_{BS,s}(t) d_s^2 N_s + \sum_{s\in\mathcal{S}} (Q_{BS,s}(t) d_s^2 - W\Delta_s) D_s^c$$

2) Departure from the MEC server:

$$D_{BS,s}(t) = \min\{Q_{BS,s}(t), N_s(t) + D_s^c(t)\}$$

Recall that the MEC server receives a type-s service request when it receives all the d_s packets corresponding to the request. Hence, there are residual packets waiting at the BS until all the packets of the corresponding service arrive. Let $c_{s,u}(t)$ be the residual packets of the type-s service from user u at the BS at slot t. Then $A_{BS,s}(t) = \sum_u \lfloor \frac{c_{s,u}(t)+D_{s,u}(t)}{d_s} \rfloor$ and $c_{s,u}(t) = c_{s,u}(t-1) + D_{s,u}(t-1) - \lfloor \frac{c_{s,u}(t-1)+D_{s,u}(t-1)}{d_s} \rfloor d_s$. Let $\Gamma(t) = (Q_{s,u}(t), Q_{BS,s}(t))$. It is easy to see that $\Gamma(t), t \geq 0$ evolves as an irreducible Markov Chain [18] under BP based drift-plus-penalty algorithm. The following theorem states the Markov chain $\Gamma(t), t \geq 0$ is irreducible and positive recurrent thereby implying that the MEC system is stable under BP based drift-plus-penalty algorithm.

Theorem 5. *The MEC system is stable under BP based drift-plus-penalty algorithm for any arrival rate λ such that $(1+\epsilon)\lambda \in \mathcal{C}$ for any $\epsilon > 0$.*

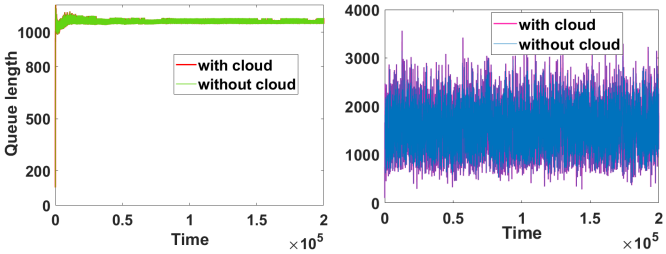
Proof: (outline) The proof follows from Lyapunov drift analysis for the Lyapunov function $L(\Gamma(t)) = \sum_s \sum_u Q_{s,u}^2(t) + \sum_s Q_{BS,s}^2(t) d_s^2$. ■

Remark 2. *If $d_s = 1$ for all s (i.e., each service has unit data size), the above algorithm is exactly same as BP-based scheduling for multihop communication networks.*

V. Numerical Results

In this section, we evaluate the performance of Algorithm 1 via simulations. Then, we compare the queue lengths and *delay cost* of Algorithm 1 and Algorithm 2. We consider 5 different services, 20 users associated with a BS. At last, we observe the evolution of queue length for different ϵ. We study two performance metrics, the aggregate queue lengths (i.e., the sum of queue lengths of 20 users at the users and the sum of queue lengths of 5 services at the MEC server) and the average *delay cost*. The MEC server has a storage capacity of 25 GB and CPU of frequency 5 GHz [6], and the cloud has a transmission rate $C_{cloud} = 100$ Mbps [19].

Service Specifications [6], [20]:



(a) virtual queue length (b) actual queue length

Fig. 2: Aggregate queue length evolution at the user

Services	Data Size (MB)	Storage (GB)	CPU cycles ($\times 10^9$)	RTT delay for computing at the cloud (s)
Face Recognition	10	10	2.5	3
Augmented Reality	10	12	3	3
Mobile cloud Gaming	8	16	3.5	2.4
Wearable sensor	1.5	10	1	1
Video editing	9	2	3	6

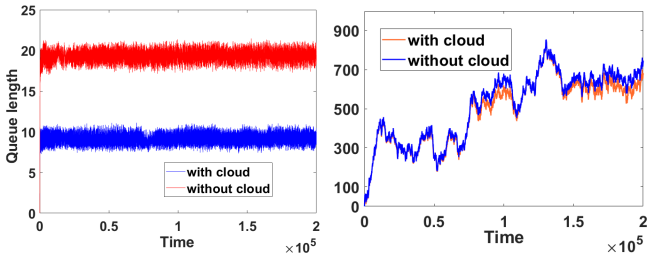
We consider each slot length is 1s, each packet size is 1 Mb, and $\epsilon = 10^{-4}$, and the transmission channel between is the Gilbert-Elliot channel with transition probability matrix $\begin{bmatrix} 0.6 & 0.4 \\ 0.5 & 0.5 \end{bmatrix}$. The uplink rate for two different channel states are 199 and 163 Mbps [6]. We assume that the arrival process is i.i.d. over the slots and in each slot number of arrivals is a Poisson random variable with a given arrival rate vector.

A. Throughput-optimality of the MEC system without the cloud

In this subsection, we demonstrate that Algorithm 1 is throughput-optimal for the MEC system excluding the cloud (i.e., $W = \infty$). For this, we consider the arrival rate per user to be 4.9896 Packets/s and the arrival rate vector for services to be $[0.396, 0.396, 0.198, 0.7920, 0.1980]$ services/s. The traffic load constitutes 99 percent of the capacity region. We execute the simulations for 200000 time slots. Figure 2 and 3 show that the queues are stable. Since the queue length at the user is in bits and at the MEC server in service, we observe the aggregate queue length in each slot at user is higher than at the MEC server.

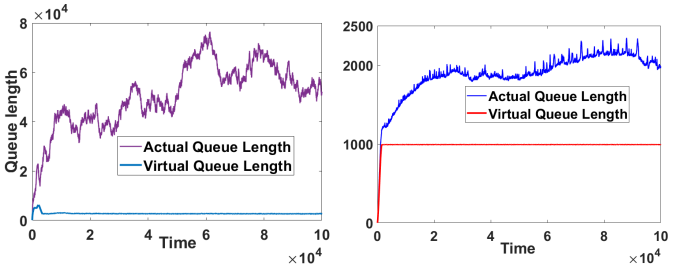
B. Throughput-optimality of the MEC system with the cloud

In this subsection, we demonstrate that Algorithm 1 is throughput-optimal for the MEC system with the cloud. For this, we consider the arrival rate per user to be 8.8308 Packets/s, the arrival rate vector for services to be $[0.594, 0.792, 0.396, 0.99, 0.396]$ per s, $W = 100$. The traffic load constitutes 99 percent of the capacity region. We execute the simulations for 100000 time slots. Figure 4 shows that the queues are stable.



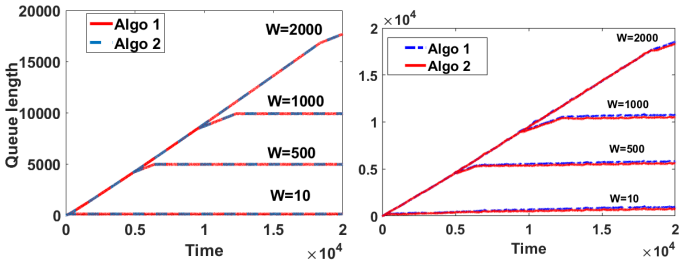
(a) virtual queue length (b) actual queue length

Fig. 3: Aggregate queue length evolution of the MEC system with and without cloud



(a) at user (b) at the MEC server

Fig. 4: Aggregate queue length evolution of the MEC system



(a) virtual queue length (b) actual queue length

Fig. 5: Aggregate queue length evolution of the MEC system for different values of W

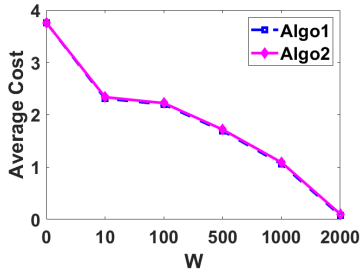


Fig. 6: Average *delay cost* (*delay for service execution at the cloud*) for different values of W

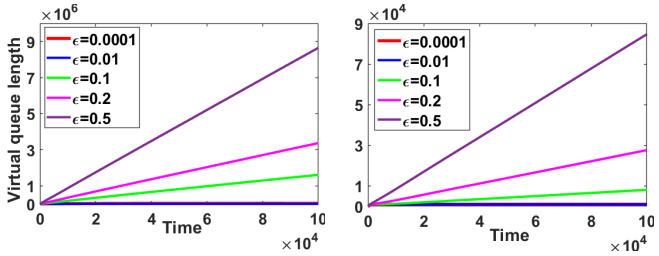
1) Comparison: We consider the same arrival rate as subsection A and compare the queue length evolution of the MEC system after adding cloud (i.e., W=1). Figures 2a and 2b show that adding the cloud to the MEC system does not affect the user queue length since it can not change the departure rate from the user queues. However, it affects the departure to the MEC server. It is clear from Figure 3a and 3b the queue lengths of actual and virtual queues reduce when we add the cloud to the MEC system. The average actual queue length (i.e., averaged over 200000 iterations) at the MEC server with and without the cloud are 489.07 and 511.41 services, respectively. The reduction of queue length is unsurprising since adding the cloud increases the overall capacity region if the MEC server acts as a bottleneck of the capacity region.

2) Effect of parameter W and comparison between Algo. 1 and 2: We consider the same arrival rate as subsection B and vary W from 0 to 2000, and run the simulations for 20000 iterations. We only observe the change in average *delay cost* and the queue lengths at the MEC server since changing W does not affect the queue lengths of the users. As We mention in the section III, observe that under Algorithm 1 increasing the parameter W increases the queue lengths in Figure 5 and decreases the average *delay cost* in Figure 6. Therefore, W acts as a trade-off parameter between the queue length and the average *delay cost*. The same phenomenon occurs under Algorithm 2 for the same reason. Figure 5b shows that the actual queue lengths are slightly higher for Algorithm 1 than for Algorithm 2. The virtual queue evolution under Algorithm 1 and Algorithm 2 are very close. But the average virtual queue length (i.e., averaged over 20000 iterations) is smaller under Algorithm 1. For example, they are 4202.8 and 4205.4 services under Algorithm 1 and 2, respectively for W=500. In contrast, the average *delay cost* is slightly lower for Algorithm 1 than for Algorithm 2, see Figure 6. Although Algorithm 2 sends the services to the cloud in a more controlled manner, the average *delay cost* is slightly higher than Algorithm 1. Therefore Algorithm 1 performs better in terms of average *delay cost*.

3) Effect of parameter ϵ: This subsection demonstrates how the parameter ϵ affects the queue lengths. The virtual queues in Figure 7 and 8 are stable for ϵ=0.01, 0.0001 and grow linearly with time for ϵ=0.1, 0.2 and 0.5. The reason is for $(1+\epsilon)\lambda \in \mathcal{C}$ for the values of ϵ=0.01, 0.0001 and $(1+\epsilon)\lambda \notin \mathcal{C}$ for the values of ϵ=0.1, 0.2 and 0.5. The actual queues are stable when the virtual queues are stable (i.e., for ϵ=0.01, 0.0001). Since larger ϵ implies more arrivals to the virtual queues where the capacity region is the same, the average queue length of the virtual queues for ϵ=0.01 is larger than ϵ=0.0001. However, the average queue length of the actual queues for ϵ=0.01 is smaller than ϵ=0.0001. The stability of the virtual queues ensures that the service rate of the virtual queues is at least $(1+\epsilon)\lambda$. Since the service rate is the same for both the virtual and actual queues, for a given arrival rate, the guaranteed service rate increases as we increase ϵ. Figure 7 and 8 show that the actual queues can be stable or unstable when the virtual queues are unstable, e.g., for ϵ=0.1, 0.2, 0.5.

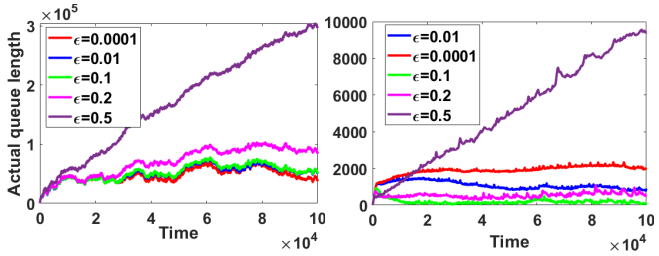
C. Comparison to BP-based drift-plus-penalty

We consider the same arrival rate as in Section V-B and plot the average queue lengths vs. costs for different W. We run the simulations for 20000 iterations. The average queue length is calculated, averaging the users' and MEC server's total queue lengths over 20000 iterations. We vary W from



(a) at user (b) at the MEC server

Fig. 7: Aggregate virtual queue length evolution of the MEC



(a) at user (b) at the MEC server

Fig. 8: Aggregate actual queue length evolution of the MEC system

10 to 5000 and from 8×10^5 to 1.2×10^7 for Algorithm 1 and BP based drift-plus-penalty algorithm, respectively. We note that Algorithm 1 offers almost same performance as BP-based drift-plus-penalty (see Figure 9). However, we need to employ different W in the two algorithms for similar queue length-cost pairs. As we demonstrated in Section V-B, Algorithms 1 and 2 offer very close performance, so we need not compare Algorithm 2 and BP-based drift-plus-penalty.

VI. Conclusions

We have proposed throughput optimal algorithms for task offloading and scheduling which can achieve average optimal delay arbitrarily closely without using cross layer communication. Theorems 2 and 4 establish that the proposed algorithms are throughput-optimal. Theorems 3 and 4 establish that the algorithms can yield average *delay costs* arbitrarily close to the optimal.

It is of interest to design a sub-optimal solution to the MEC configuration problem that can provide throughput-optimality or can achieve a substantial fraction of the capacity region. It also remains to investigate problems where the services require processing times of more than one slot and experience dynamic delay costs when executed at the cloud.

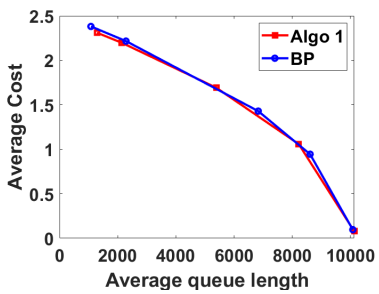


Fig. 9: Average *delay cost* (*delay for service execution at the cloud*) vs queue length for different values of W

References

[1] Z. Jiang and S. Mao, "Energy delay tradeoff in cloud offloading for multi-core mobile devices," *IEEE Access*, vol. 3, pp. 2306–2316, 2015.

[2] A. Katal, S. Dahiya, and T. Choudhury, "Energy efficiency in cloud computing data centers: a survey on software technologies," *Cluster Computing*, pp. 1–31, 2022.

[3] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, (New York, NY, USA), p. 49–62, Association for Computing Machinery, 2010.

[4] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2131–2165, 2021.

[5] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 207–215, 2018.

[6] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in mec networks with storage, computation, and communication constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1047–1060, 2020.

[7] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, 2016.

[8] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Mobile edge computing network control: Tradeoff between delay and cost," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1–6, 2020.

[9] S. Liu, E. Ekici, and L. Ying, "Scheduling in multihop wireless networks without back-pressure," *IEEE/ACM Transactions on Networking*, vol. 22, no. 5, pp. 1477–1488, 2014.

[10] B. Ji, C. Joo, and N. Shroff, "Throughput-optimal scheduling in multihop wireless networks without per-flow information," *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, pp. 634–647, 2013.

[11] M. Bramson, B. D'Auria, and N. Walton, "Stability and instability of the maxweight policy," *Mathematics of Operations Research*, vol. 46, no. 4, pp. 1611–1638, 2021.

[12] R. Srikant and L. Ying, *Communication networks: an optimization, control, and stochastic networks perspective*. Cambridge University Press, 2013.

[13] A. Koley and C. Singh, "Throughput and delay optimal scheduling for multi-access edge computing without using cross-layer communication." https://www.dropbox.com/sh/9oeq0v1wsbtk5qh/AAAMT4Aa4IdKEKoUvRz7Z76na?dl=0, 2023.

[14] J. G. Dai, "On positive harris recurrence of multiclass queueing networks: a unified approach via fluid limit models," *The Annals of Applied Probability*, vol. 5, no. 1, pp. 49–77, 1995.

[15] M. Andrews, K. Kumaran, K. Ramanan, A. Stolyar, R. Vijayakumar, and P. Whiting, "Scheduling in a queuing system with asynchronously varying service rates," *Probability in the Engineering and Informational Sciences*, vol. 18, no. 2, p. 191–217, 2004.

[16] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[17] H. Kellerer, U. Pferschy, D. Pisinger, H. Kellerer, U. Pferschy, and D. Pisinger, "The bounded knapsack problem," *Knapsack Problems*, pp. 185–209, 2004.

[18] S. T. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *2012 Proceedings IEEE INFOCOM*, pp. 702–710, 2012.

[19] H. T. Malazi and S. Clarke, "Distributed service placement and workload orchestration in a multi-access edge computing environment," in *2021 IEEE International Conference on Services Computing (SCC)*, pp. 241–251, 2021.

[20] R. Montella, S. Kosta, D. Oro, J. Vera, C. Fernández, C. Palmieri, D. Di Luccio, G. Giunta, M. Lapegna, and G. Laccetti, "Accelerating linux and android applications on low-power devices through remote gpgpu offloading," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 24, p. e4286, 2017.