Preemptive Mobile Computation Offloading with Hard Deadlines and Concurrent Local Execution

Peyvand Teymoori, Arvin Hekmati, Terence D. Todd and Dongmei Zhao
Department of Electrical and Computer Engineering
McMaster University
Hamilton, Ontario, CANADA
Email: {teymoorp,ahekmati,todd,dzhao}@mcmaster.ca

George Karakostas
Department of Computing and Software
McMaster University
Hamilton, Ontario, CANADA
Email: karakos@mcmaster.ca

***Abstract*—This paper considers Preemptive Mobile Computation Offloading when concurrent local execution (CLE) is used to guarantee task execution time constraints. By allowing simultaneous local and remote execution, CLE ensures that job deadlines are always satisfied in the face of unforeseen wireless channel conditions. In the preemptive offloading case, at the start of each time slot, a decision is made to either continue or temporarily interrupt the offload. This mechanism allows the system to adapt when channel conditions change. The paper considers the case for homogeneous Markovian wireless channels. Using Markovian decision process stopping theory, an online energy-optimal computation offloading algorithm is formulated for preemptive offloading, referred to as Optimal Preemptive Offloading (OPO). Since the computational complexity of OPO can be prohibitive, the paper introduces three computationally efficient techniques motivated by OPO: *Water-Filling*, *Water-Filling with Scheduling*, and *Generalized Water-Filling*. For each method, two variations are considered. The first (*Equ*) uses the equilibrium channel state probabilities in offloading decision calculations, and the second (*Exp*) uses Markovian transition matrix exponentiation. This results in six algorithms with a wide variety of energy performance and computational complexity. Performance of the algorithms is compared that shows the trade-offs between complexity and mobile energy saving performance.**

***Index Terms*—Cloud computing, mobile computation offloading, energy efficiency, mobile task execution performance, hard job deadline constraints.**

I. Introduction

Mobile computation offloading (MCO) can be used to improve application performance and reduce mobile energy use. This is done by having the mobile device offload local task execution to a remote cloud server, rather than running the task on the device itself. Wireless communications is typically used by the mobile device to upload the task/data so that remote execution is possible. There is a large literature that has studied various issues dealing with mobile computation offloading [1] [2] [3] [4] [5].

In computation offloading, mobile execution energy can be decreased, but is offset by an increase in the communication energy needed to interact with the cloud server. Offloading will also incur additional latency that would not exist otherwise due to the time needed to exchange application data with the server. This latency may be partially compensated for by a faster task execution time at the cloud server. The basic tradeoffs involving these attributes and how they relate to the decision to offload task execution have been studied extensively, for example cf. [6] [7] [8], and the references therein.

The decision to offload task execution is more complicated when the mobile device interacts with the cloud over stochastic transmission channels. This issue was studied in [9], where an energy model was proposed that considers both computation and energy components, using the statistics of the wireless channel. The channel was assumed to remain constant throughout the computation offload, at the state encountered by the mobile device at the start of the offload. In more general situations, the wireless channel may randomly evolve during the computation offload, which further complicates the decision to offload or to execute a given task locally. This is the environment that is considered in this paper.

When task execution times must satisfy hard time constraint deadlines, computation offloading over time-varying stochastic channels is particularly difficult. The requirement for execution time constraints was discussed in [10], which identified this as an important criterion for many interactive applications. This paper also discussed the difficulty of achieving this under random wireless channel conditions. Computation offloading was addressed in [11] assuming random wireless channels, but without considering hard task execution deadlines. Dynamic programming was used to optimize computation offloading decisions in [12], but task execution time deadlines were not considered. Reference [13] used mobile CPU frequency scheduling and transmit power control to compensate for random wireless channel effects, so that remote task offload latencies can be controlled. This approach cannot always ensure that task execution deadlines are met, but a parameter was introduced that controls the probability that task deadlines can be violated.

Reference [14] proposed an on-line mobile computation offloading algorithm, OnOPT, which was shown to be energy optimal and satisfies hard deadline constraints. Deadline constraints are satisfied by using Concurrent Local Execution (CLE), where local task execution may be initiated even if remote offloading is in progress. This is to ensure that task deadlines are guaranteed in the face of any unforeseen channel conditions. References [15], [16] proposed an on-line mobile computation offloading algorithm, MultiOPT, that was shown to be energy optimal and satisfies hard deadline constraints for the case of splitting the offloading data into two or more parts of known bit-sizes.

Our paper uses *concurrent local execution (CLE)* and considers Preemptive Mobile Computation Offloading, referred to as *preemptive offloading* for short. In the preemptive case, the algorithm makes separate upload initiation time decisions at the start of *every* time slot. Based on its inputs, it decides whether to use the current time slot to continue the upload or defer uploading to future time slots. Preemptive offloading can be used to reduce mobile energy use when wireless channel conditions change during the computation offload. These decisions are made such that the system always satisfies the given hard task execution time constraint using concurrent local execution offloading. The objective is to minimize mobile device energy consumption, subject to this constraint. It is very easy to illustrate the value of preemption in computation offloading. For example, consider a mobile device that has a large job offload over a 2-state Markovian channel (i.e., a Gilbert-Elliot channel), and toggles between good and bad channel states with the same mean time in each state, and at a rate much smaller than the upload time. Then the offload energy needed for the non-preemptive case would be roughly twice that of using preemption. The statistics of the channel can easily be such that the advantage to the preemptive case is far greater.

The paper considers the homogeneous Markovian channel case, which is commonly used to model random wireless channel conditions. Under this assumption, computation offloading algorithms are introduced for preemptive offloading. The paper shows that the proposed algorithm is energy optimal and is referred to as *Optimal Preemptive Offloading (OPO)*. This algorithm is optimal in the sense that no other online computation offloading algorithm can achieve a lower mean mobile device energy consumption. The energy optimality of this algorithm is shown by creating a time-dilated absorbing Markov processes and using optimal Markovian stopping theory. More specifically, it is shown that a dynamic programming approach can be used to compute the expected energy cost in case offloading occurs or not in the current time slot, and the algorithm bases its (optimal) decision on these costs.

Since the computational complexity of OPO can be significant, the paper introduces three computationally efficient techniques, motivated by OPO, namely, *Water-Filling*, *Water-Filling with Scheduling*, and *Generalized Water-Filling*. These methods operate by first identifying a target set of future time slot types to use for the offload. This set is defined based on predicted bit rates using the Markovian channel statistics. Time slots are then assigned using "water-filling", which prioritizes those with higher expected bit rates. The algorithms vary based on how the computations are performed. For each method, two variations are considered. The first (*Equ*) uses the equilibrium channel state probabilities in its offloading decision calculations. The second variant (*Exp*), while more accurate, uses Markovian transition matrix exponentiation, which is more computationally expensive. This results in six algorithms, namely, *Water-Filling with Equilibrium (WF-Equ)*, *Water-Filling with Exponentiation (WF-Exp)*, *Water-Filling with Equilibrium and Scheduling (WF-Equ-Sch)*, *Water-Filling with Exponentiation and Scheduling (WF-Exp-Sch)*, *Generalized Water-Filling with Equilibrium (Gen-WF-Equ)* and *Generalized Water-Filling with Exponentiation (Gen-WF-Exp)*. The performance of the proposed algorithms is compared on Markovian channels with different characteristics, in order to show the tradeoffs between complexity and mobile energy saving performance.

The main contributions of the paper are summarized as follows:

- Preemption is introduced to concurrent local execution, used in mobile computation offloading in order to ensure that hard job execution deadlines are satisfied. Compared to previous work where offloading occurs using known job offload parts [14],[15],[16], allowing preemption results in a much more complex problem formulation. This happens because the number and size of the job pieces offloaded are not known in advance and must be determined by the online offloading algorithm. Due to this complexity, it is not feasible to run the optimal online decision algorithm in real time, and this motivates the use of the proposed computationally efficient techniques.
- An online offloading decision algorithm, i.e., Optimal Preemptive Offloading (OPO), is introduced. It is shown that OPO satisfies hard deadline application constraints, and also achieves the minimum mean mobile device energy possible for homogeneous Markovian wireless channels.
- An integer program (IP) is formulated that provides a lower bound on mobile device energy. This calculation is used for comparisons with online algorithms in the results section.
- The paper introduces three computationally efficient techniques based on OPO: *Water-Filling*, *Water-Filling with Scheduling*, and *Generalized Water-Filling*. The algorithms vary based on how the reduced offloading decision computations are performed. For each method, two variations are introduced. The first (*Equ*) uses the equilibrium channel state probabilities in its offloading decision calculations and the second (*Exp*) uses Markovian transition matrix exponentiation. This gives six algorithms: *Water-Filling with Equilibrium (WF-Equ)*, *Water-Filling with Exponentiation (WF-Exp)*, *Water-Filling with Equilibrium and Scheduling (WF-Equ-Sch)*, *Water-Filling with Exponentiation and Scheduling (WF-Exp-Sch)*, *Generalized Water-Filling with Equilibrium (Gen-WF-Equ)* and *Generalized Water-Filling with Exponentiation (Gen-WF-Exp)*. The algorithms all employ concurrent local execution so that task execution time constraints are

always satisfied.

- Performance results are presented that show the various complexity and energy performance tradeoffs for the proposed algorithms.

The rest of this paper is organized as follows. In Section II, system modelling assumptions are presented that include both remote and local execution where hard task execution deadlines are ensured using CLE. In Section III, an offline lower bound on energy consumption is derived, which is compared to the proposed computation offloading algorithms in the results section. Following this, in Section IV the Markovian channel model is discussed and how it is used to define an absorbing Markov chain that permits us to show the energy optimality of the proposed algorithm OPO, introduced in Section IV. Section V then introduces the online algorithms motivated by OPO, but with varying degrees of running time reduction. In Section VI, results are presented that compare the various algorithms and show the tradeoffs between computational complexity and mobile energy savings. Finally, our conclusions are presented in Section VII.

II. System Model

We consider the execution of single computational tasks (jobs) generated by a mobile device, either locally (by the device itself), or by offloading them on a remote cloud server, through a wireless transmission channel. We allow a job to be uploaded preemptively to the server in instalments, i.e. a piece of the job can be uploaded at each time slot, if the upload scheduler decides to do so, i.e., *preemption* is allowed. We propose a model of *concurrent local execution (CLE)*, i.e., we allow for the overlapping of offloading and local execution, so that the hard job deadline is always met by at least the local execution. In case offloading results become available before local execution finishes (or even starts), the latter is automatically aborted. Similarly, if the deadline is reached while remote execution is not finished, the latter is aborted. The objective of the proposed algorithms is to minimize the mean energy consumption of the mobile device.

Note that time is taken to be discrete, i.e., quantized into equal length *time slots*. The time slot duration is defined so as to accommodate the channel propagation model discussed in Section IV, and may contain multiple packet transmission times on the channel. Each job to be executed is characterized by the following:

t_R:	*Release time* of the job, i.e., the time when the job is ready to start execution, either locally or via offloading. This is marked on the left side of Figure 1. For convenience, we will assume that $t_R = 1$.
t_D:	*Hard deadline* of the job, i.e., the job execution results *must* be available at the mobile device by time t_D. This is shown on the right side of Figure 1, where $T_D = t_D - t_R + 1$ is the maximum number of time slots available for completing the job.
S_{up}:	Number of bits transmitted through the uplink channel when uploading the job to the cloud

Preemptive offloading is shown in Figure 1. Uploading happens at time slots $t_{o_1}, t_{o_2}, \ldots, t_{o_n}$, and T_{W_i} is the time period between t_{o_i} and $t_{o_{i+1}}$. T_{exec}, T_{down}, T_L are the remote execution, result downloading, and local execution times, respectively.

As in many of the references, we assume that the current state of the channel can always be determined. This information can be learned in a variety of ways, such as via a short handshake with the basestation at the start of the time slot.

A. Local Execution

As in [14], it is assumed that the energy cost E_L and time T_L needed to execute a job locally is known at the job release time t_R. While this may not always be the case, this assumption is often true and has been made in many computational offloading studies, e.g., [6], [17], [18].

If the computation offloading algorithm elects to execute the job locally, we must ensure that the job deadline is always satisfied. Therefore, local execution must start no later than

$$t_L = t_D - T_L + 1,$$

i.e., local execution must start T_L time slots prior to the job deadline, if remote execution results have not arrived by then (see Figure 1). Note that the satisfaction of hard job execution deadline constraints would be problematic if there were uncontrolled preemption in the mobile device itself. For this reason, we have assumed that the features normally associated with a real-time operating system are in place so that the job execution is assigned a known fraction of the local processor. In this way, T_L can be calculated when the task is released. Hence, according to the CLE model we are proposing, there may be overlapping of the local and remote executions of the task, but the hard job deadline is *always* respected, even if there is channel contention or extended channel outages. We assume that a fraction of the local CPU is dedicated for processing the task so that the local processing speed is fixed and known, and T_L can be calculated when the task is released.

In the mobile energy consumption formulations described above and in the next section, we have chosen not to include the local energy requirements needed to execute the online scheduling algorithms. This is consistent with the common assumption that the energy needed for offloading the job data is large, by comparison. However, in Section VI, graphs of the relative running times of the algorithms have been included so that this component can be accounted for, if required.

B. Remote Execution

In the case of offloading a job, we will assume that, upon its release, the job is assigned an execution time T_{exec} by the cloud server, which is communicated to the mobile device (or is prescribed by, say, the contractual agreement between the user of the device and the cloud server operator). In addition, we assume that the user has been allocated capacity (such as recurring time slots) until the offload has completed. These assumptions are common in previous work on offloading, e.g., [6] [17] [18]. Following Figure 1, the total offloading time T_{off} is given by

$$T_{off} = t_{o_n} - t_{o_1} + 1 + T_{exec} + T_{down}.$$

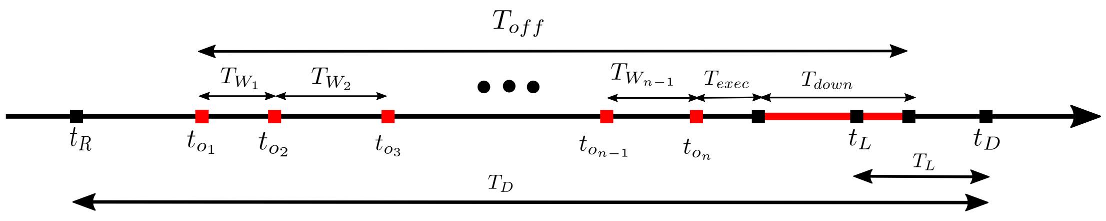


Fig. 1: Preemptive offloading.

In what follows, we define

$$T_{rest} = T_{exec} + T_{down}$$

It is assumed that the channel uses bit rate adaptation to accommodate random variations in channel conditions. As a result, T_{off} is a random variable, dependent on the evolution of the uplink channel state as a given upload occurs. In what follows, it is assumed that the channel state can be modelled as a homogeneous discrete-time Markov process.

III. Offline bound

In this section, an offline lower bound on mobile device energy is derived. This bound is used in Section VI for performance comparisons with various online computation offloading algorithms. Since the bound is offline, we assume that the wireless channel states are known beforehand, i.e., we know the bit rate (in bits per time slot) at all times $1 \le t \le t_D$ (recall that we have set $t_R = 1$). When a job is released, the bound chooses the job offload times so that its deadline is met and total energy is minimized.

Let x_i be the decision of offloading at time slot t_i, i.e., $x_i = 0$ when we decide to offload at time slot i, and $x_i = 1$ otherwise. Let t_f be the upload finishing time. B_i is the bit rate at time slot i, and E_{tr} is the energy cost per time slot for channel transmitting. The optimal x_i's are found by first solving the following set of IPs (one for each possible finishing time t_f), and then output the minimum amongst them and E_L (if E_L is the minimum, then the optimal solution is to not offload at all).

- $t_f < t_L - T_{rest}$

$$\min_{x_1,\cdots,x_{t_f}} E_{tr}\sum_{i=1}^{t_f} x_i + T_{down}E_{rc} \tag{1}$$

$$\text{s.t.} \sum_{i=1}^{t_f} B_i x_i \ge S_{up} \tag{2}$$

$$x_{t_i} \in \{0,1\} \text{ for } i = 1,2,\cdots,t_f - 1 \tag{3}$$

$$x_{t_f} = 1 \tag{4}$$

- $t_L - T_{rest} \le t_f \le t_D$

$$\min_{x_1,\cdots,x_{t_f}} E_{tr}\sum_{i=1}^{t_f} x_i + \frac{t_f + T_{rest} - t_L + 1}{T_L} E_L + T_{down}E_{rc} \tag{5}$$

$$\text{s.t.} \sum_{i=1}^{t_f} B_i x_i \ge S_{up} \tag{6}$$

$$x_{t_i} \in \{0,1\} \text{ for } i = 1,2,\cdots,t_f - 1 \tag{7}$$

$$x_{t_f} = 1 \tag{8}$$

The first case corresponds to finishing offloading before local execution begins, and the second to finishing offloading after local execution begins. The first term in (1) and (5) is the uploading energy cost. The second term in (5) is the portion of the local energy cost we incur if offloading finishes at t_f. Constraints (2) and (6) ensure all job bits are offloaded. Constraints (4) and (8) ensure that uploading finishes at t_f.

IV. Optimal Algorithm for Preemptive Offloading

In this section, we develop the online algorithm OPO (Optimal Preemptive Offloading), and prove its optimality. In order to simplify our exposition of the algorithm, we will assume that all offloading deadlines, job sizes (in bits), and energy costs are related only to job uploading, i.e., we assume that T_{exec}, T_{down} are known and already incorporated in the energy costs and the job deadline. Given the ensuing results, adding the effects of T_{exec} and T_{down} is straightforward.

As mentioned above, we assume that there is a known channel state Markov chain (CSMC), i.e., the channel conditions evolve from one time slot to the next according to a homogeneous finite state Markov chain. Each state in the CSMC has an associated bit rate that gives the number of bits per time slot that can be uploaded in that state. The CSMC transition matrix is defined as $\mathbb{P} = [P_{i,j}]$, where $P_{i,j}$ is the probability of transitioning to channel state j in the next time slot, given that the channel is currently in state i. As defined, CSMC is memoryless, but in what follows we will need to incorporate time in it, the already offloaded bits, and the decision of uploading or not at every time slot. Therefore, we construct a new Markov decision process, the *time-dilated Markov process (TDMP)* as follows: For every time $1 \le t \le t_D + 1$, we define a set of states (X_t, S_t), where X_t is a channel state and $0 \le S_t \le S_{up}$. We can think of the states as arranged in layers for $t = 1, 2, \ldots, t_D + 1$. The set of actions contains two actions, a_0, a_1, corresponding to not offloading, or offloading, respectively. Figure 2 shows the initial layers for a TDMP for a Gilbert-Elliot channel with two states, G and B, with bitrates B_g, B_b, respectively. A state (X_t, S_t) branches to a_0 and a_1, and then a_0 branches to states (X_{t+1}, S_t) with probabilities $P_{X_t,X_{t+1}}$, and a_1 to states $(X_{t+1}, S_t + \min\{B_{X_t}, S_{up} - S_t\})$ with the same probabilities, where B_{X_t} is the bit rate of channel state X_t. States of the form (X_t, S_{up}) lead only to action a_0 (no offloading). At layer $t = 1$ there is only one state $(X_1, 0)$, where X_1 is the initial channel state, while all states at layer $t_D + 1$ are absorbing.

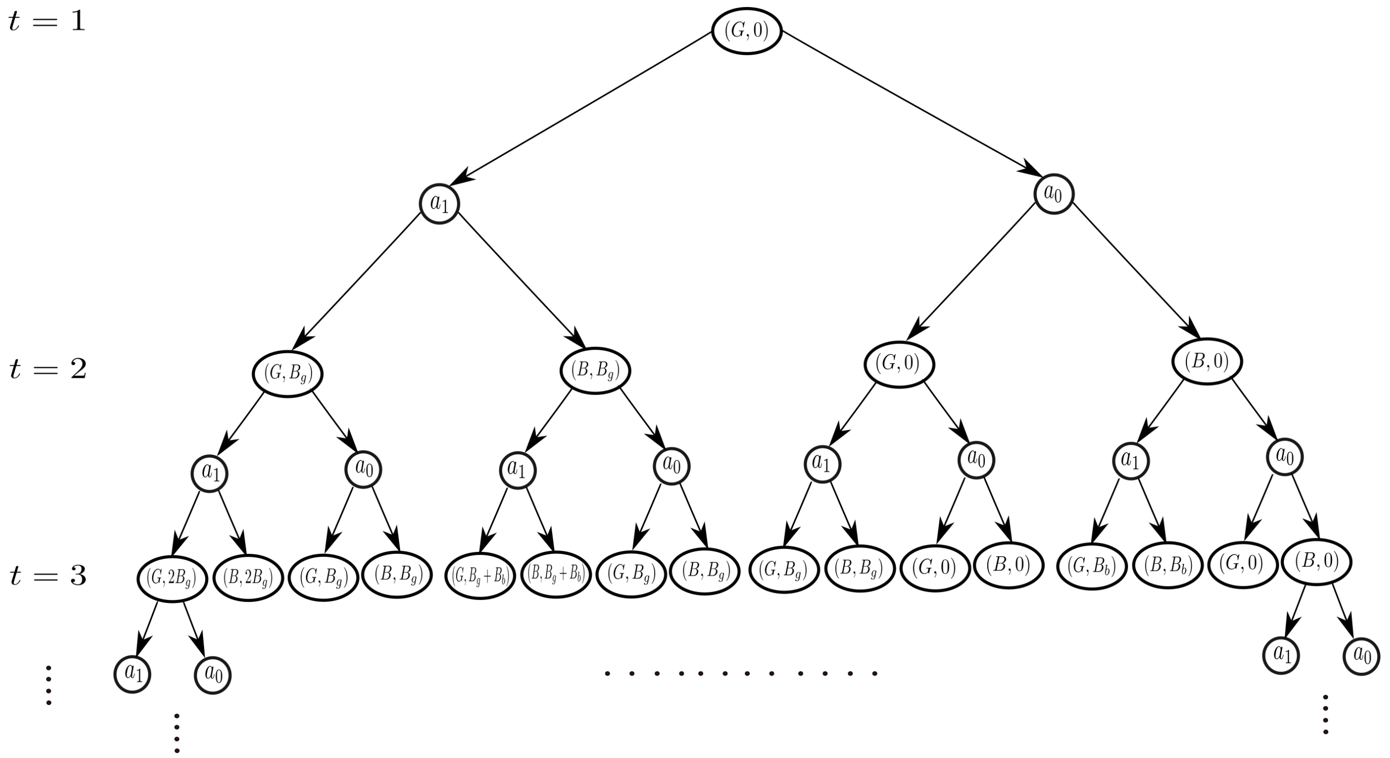


Fig. 2: Time dilated absorbing Markov process (TDMP) for a two-state (G, B) channel.

The energy cost when offloading in time slot t is given by:

$$E_{off}(t) = \begin{cases} E_{tr}, & t_R \le t \le t_D \\ 0, & t < t_R \text{ or } t > t_D \end{cases} \tag{9}$$

The energy cost for the local execution between the previous offloading time slot t_{prev} and the current time slot t (note that $t > t_{prev}$) is given by:

$$E_L(t_{prev}, t) = \begin{cases} 0, & t < t_L \text{ or } t_{prev} \ge t_D \\ \frac{\min\{t, t_D\} - \max\{t_{prev}+1, t_L\}+1}{T_L} E_L, & \text{o/w} \end{cases} \tag{10}$$

The expected energy cost of offloading, when offloading occurs at time t in TDMP state X_t, the last offloading time slot is t_{prev}, and S_t bits have already uploaded, is

$$g(X_t, S_t) = E_{off}(t) + E_L(t_{prev}, t) + \sum_{X_{t+1} \in \mathcal{M}} Pr[X_{t+1}|X_t] V(t, X_{t+1}, S_t + B_{X_t}). \tag{11}$$

Note that the definition of g includes energy costs due to local execution that were accumulated in the time slots between the last offloading at t_{prev} and the current one at t.

For every time slot t and state X_t, and with the last uploading time slot being t_{prev}, we define the expected cost $V(t_{prev}, X_t, S_t)$ recursively in (12). In (12), we have

$$E[V(t_{prev}, X_{t+1}, S_t)|X_t] = \sum_{X_{t+1} \in \mathcal{T}} Pr[X_{t+1}|X_t] V(t_{prev}, X_{t+1}, S_t),$$

and $\mathcal{T}$ is the set of states reachable after running the channel for $t+1$ time slots. The first case has an expected cost of 0, since there is no more offloading. The second case incurs only local energy consumption, because the time t is beyond the deadline t_D, and, therefore, offloading doesn't happen. The third case assigns as the expected cost the minimum between the expected cost of offloading at t, and the expected cost of postponing that decision to time slot $t+1$.

The formal definition of algorithm OPO is Algorithm 1. Recall that, for simplicity, we have set $t_R = 1$. A high-level description of the algorithm is as follows: At time slot $t = 1$ the job is released. At each time slot t, and with S_t bits already offloaded in past offloading time slots, the algorithm considers the TDMP, in order to determine the expected cost of the whole remaining offloading process, if uploading happens at time slot t. If that cost is less than the expected offloading cost when the algorithm waits one more time slot, then $t^* = t$ (t becomes an offloading time slot, with bit rate B_{X_t}, and $S_{t+1} = S_t + B_{X_t}$); otherwise, the algorithm postpones its

Algorithm 1 OPO (Optimal Preemptive Offloading)

```
Input: Local execution starting time $t_L$, local execution energy $E_L$, job deadline $t_D$.
1: for all $t = 1, \ldots, t_D$ do
2:     if uploading the whole job is finished then
3:         Break
4:     end if
5:     if min in (12) is $g$ then
6:         upload at time slot $t$
7:     end if
8: end for
```

$$V(t_{prev}, X_t, S_t) = \begin{cases} 0, & t > t_{prev} \geq t_D \text{ or } S_t \geq S_{up} \\ \frac{t_D - \max\{t_{prev}+1, t_L\}+1}{T_L} E_L, & t > t_D > t_{prev} \text{ and } S_t < S_{up} \\ \min\{g(X_t, S_t), E[V(t_{prev}, X_{t+1}, S_t)|X_t]\}, & t_D \geq t > t_{prev} \text{ and } S_t < S_{up} \end{cases} \quad (12)$$

decision to time slot $t+1$, and $S_{t+1} = S_t$. The algorithm repeats the same decision process at every time slot (using the TDMP corresponding to the bits offloaded so far, to compute expected costs), in order to determine the sequence of optimal offloading time slots $\{t_1^*, t_2^*, t_3^*, \ldots\}$. $V(t_{prev}, X_t, S_t)$ in Line 5 can be computed using Dynamic Programming (DP).

We use the theory of optimal stopping for Markov decision processes [19], in order to show that it achieves the optimal expected energy for the mobile device, i.e., no other online computation offloading algorithm can achieve a lower mean mobile device energy consumption. This is done by proving that at any time slot t, the algorithm makes exactly the same decision as an algorithm that has the ability, starting from t, to decide *all* future offloading decisions that minimize the expected energy cost. The high level idea of the proof of optimality is as follows: Algorithm OPO is an *on-line* algorithm that runs at each time slot t. At this time slot t, an expected energy consumption minimization problem can be defined, which computes *all* optimal offloading time slots in the time period from t to deadline t_D, given the last offloading time slot t_{prev} and the number of job bits S_t that have already been offloaded. This minimization problem is defined recursively, by calculating the expected energy cost of *future optimal* offloading decisions for each possible next offloading time $t, t+1, t+2, \ldots, t_D+1$; let $t^* \geq t$ be the next offloading time which achieves the minimum expected energy. We will prove that if $t^* > t$, algorithm OPO also decides to *not* offload at t, and if $t^* = t$ algorithm OPO also decides to offload at t. Note that t^* is the first of a series of optimal offloading time slot decisions that minimize the expected energy cost *given our current knowledge of the channel*. At every time slot, the latter changes (the new channel state is revealed), algorithm OPO has to make a new decision, and a new minimization problem is defined.

The minimization problem is formulated recursively as in (13), where $\mathcal{S}$ is the set of states reachable after running the channel for t^* time slots, $\mathcal{M}$ is the set of states the channel can transit to from X_{t^*}, $B_{X_{t^*}}$ is the bit rate of the state X_{t^*}, and $v(t^*, X_{t^*+1}, S_t + B_{X_{t^*}})$ is the optimal expected energy cost for the rest of the offloading, when the algorithm decides to upload $B_{X_{t^*}}$ bits at time t^*. We set $S_0 = 0$. Note that problem (13) is defined only for $t > t_{prev}$, i.e., after the last offloading time slot. In order to prove the optimality of on-line algorithm OPO, we prove that, for all $i = 1, 2, \ldots$, when algorithm OPO offloads for the i-th time, i.e., $V(t_{i-1}, X_t, S_t) = g(X_t, S_t)$ in Line 5, the maximization problem optimal solution also offloads, i.e., $t_i^* = t$.

Theorem 1. *The sequence of optimal times $\{t_0^* = 0, t_1^*, t_2^*, \ldots\}$ for uploading satisfies*

$$t_i^* = \underset{t_{i-1}^* < t \leq t_D}{\arg\min} \{V(t_{i-1}^*, X_t, S_t) = g(X_t, S_t)\},$$

$$i = 1, 2, \ldots, last$$

Proof: We prove the theorem by induction on i. The base case of $i = 0$ is trivially true. We assume that it is true up to $i = k-1$, i.e., the $k-1$ previous offloading decision times of OPO coincide with the first offloads $t_0^*, t_1^*, \ldots, t_{k-1}^*$ of the maximization problems defined at time slots $t_0^*, t_1^*, \ldots, t_{k-1}^*$. We prove the case of t_k^*. In what follows, whenever definitions (11), (12) are used, $t_{prev} := t_{k-1}^*$ (since the immediately previous offloading time for OPO is t_{k-1}^*, by the inductive hypothesis).

First, using (reverse) induction on t, S_t, and given t_{k-1}^*, S_{up}, we show the following:

$$\forall t > t_{k-1}^*, S_t \leq S_{up} : v(t_{k-1}^*, X_t, S_t) = V(t_{k-1}^*, X_t, S_t).$$

The base case of $t > t_D$ and $S_t \leq S_{up}$ is obviously true. Assuming that the equation holds for all time values $t+1, t+2, \ldots$ and all size values $0 \leq S_{t+1}, S_{t+2}, \ldots \leq S_{up}$, we can show that it is also true for time t and all $0 \leq S_t \leq S_{up}$, by applying Theorem 1.7 in [19].

Then $v(t_k^*, X_{t_k^*+1}, S_t + B_{X_{t_k^*}})$ in the RHS of (13) can be replaced by $V(t_k^*, X_{t_k^*+1}, S_t + B_{X_{t_k^*}})$, to get the RHS of (14). By this substitution, the original maximization problem (13) is no longer recursive (i.e., dependent on the future v values), but is transformed to an equivalent minimization problem (14), that is a function of (computable, using DP) V. Then, the definition (11) of $g(X_t, S_t)$ implies that the optimal solution of (14) (which is also the optimal solution for problem (13)) can be obtained at any time slot t by performing the test of Line 5 in OPO, which is the property in the theorem statement. ■

Compared to the previous work of [14],[15],[16], which studied the same problem when offloading is done in a *predetermined* number of job pieces, each with *known* bit size, preemption changes the nature of the problem significantly, since the number and sizes of the offloaded job pieces are initially *unknown* (note that the upper bound $last$ for i in the statement of Theorem 1 is unknown). This means that, in preemption, the number of offloads is implicitly a decision variable, together with the exact offloading times. The crucial idea of the analysis above is that by allowing the DP to store some extra information (the number of bits that have already been offloaded), the recursive definition of the minimization problem (13) and expected cost (12) do not need to know the number of offloads. Therefore, although the optimal algorithm and its analysis look similar to those in [14] and [15], preemption requires a more complicated (and computationally costly) treatment, both in its computations and its analysis (e.g., note the double induction needed in the proof of Theorem 1).

$$v(t_{prev}, X_t, S_t) = \begin{cases} 0, & t > t_{prev} \geq t_D \text{ or } S_t \geq S_{up} \\ \min\limits_{t \leq t^* \leq t_D+1} \Bigg\{ \sum\limits_{X_{t^*} \in \mathcal{S}} Pr[X_{t^*}|X_t] \Big(E_{off}(t^*) + E_L(t_{prev}, t^*) + & \\ \sum\limits_{X_{t^*+1} \in \mathcal{M}} Pr[X_{t^*+1}|X_{t^*}] v(t^*, X_{t^*+1}, S_t + B_{X_{t^*}}) \Big) \Bigg\} & t_{prev} < t \leq t_D \text{ and } S_t < S_{up} \end{cases} \tag{13}$$

$$v(t_{prev}, X_t, S_t) = \begin{cases} 0, & t > t_{prev} \geq t_D \text{ or } S_t \geq S_{up} \\ \min\limits_{t \leq t^* \leq t_D+1} \Bigg\{ \sum\limits_{X_{t^*} \in \mathcal{S}} Pr[X_{t^*}|X_t] \Big(E_{off}(t^*) + E_L(t_{prev}, t^*) + & \\ \sum\limits_{X_{t^*+1} \in \mathcal{M}} Pr[X_{t^*+1}|X_{t^*}] V(t^*, X_{t^*+1}, S_t + B_{X_{t^*}}) \Big) \Bigg\} & t_{prev} < t \leq t_D \text{ and } S_t < S_{up} \end{cases} \tag{14}$$

V. Practical heuristics

Although algorithm OPO is provably optimal, its running time is of order $\Theta(S_{up}^{f(S_{up})})$, for some polynomial function f. Even when we consider the simple two-state Gilbert-Elliot Markovian channel, T_D is not much larger than $\frac{S_{up}}{B_{min}}$ (otherwise the optimal algorithms always offload when at high bit-rate states), and even when S_{up} is relatively small. Hence, good heuristics that may be suboptimal, but run fast enough to be used on-line, have to be developed.

We present three such heuristics, motivated by the optimal OPO algorithm. We observe that the prohibitively slow step in Algorithm 1 is line 5, where the DP calculation of g is performed at every time slot, in order to compute the exact expected energy consumption for offloading the remaining task bits. Our heuristics will maintain the flexibility allowed by preemption, by also running at every time slot. But they replace the costly calculation of line 5 in OPO with an approximation of the expected energy cost, using either the *equilibrium* or *invariant* probabilities π, i.e., the solution to equation $\pi P = \pi$, where P is the transition matrix of the MC (e.g., see lines 27-30 in Algorithm 2), or transition matrix exponentiation (e.g., see lines 32-33 in Algorithm 3). Hence, for each heuristic, there are two variations: the first (Equ) uses equilibrium probabilities, and the second (Exp), which is more accurate but also more computationally-intensive, uses transition matrix exponentiation.

A. Water-Filling

The basic idea of this algorithm is the computation of a "most efficient" set $\mathcal{F}$ of channel states in a greedy "water-filling" fashion, and offloading only when the current channel state is in this set, provided that offloading is beneficial at all. In accordance with the variations discussed above, there are two water-filling variants, as follows.

- **Water-Filling with Equilibrium (WF-Equ) (Algorithm 2)**: In this variation, the algorithm is given the equilibrium probabilities π (e.g., computed in preprocessing). As described above, these probabilities are used in order to compute a "most efficient" set $\mathcal{F}$ of channel states. This set is used in a greedy "water-filling" fashion, i.e., offloading only when the algorithm finds itself in this set (if it decides to offload at all). More specifically, the channel states are ordered from the highest to the lowest bit rate. Let m be the state with the highest bit rate from the state space M. If there is nothing remaining to offload, the algorithm terminates (lines 2-4). Initially $\mathcal{F} = \{m\}$ (line 5). Lines 7-20 calculate the expected finishing time to offload the remaining job, if only states in the current $\mathcal{F}$ are used, with an average bit rate $B_{avg} = \sum_{i \in \mathcal{F}} \pi[i] Br[i]$ ($Br[i]$ is the state i bitrate). As long as the states in $\mathcal{F}$ are not sufficient to meet the deadline, the next highest bit-rate state is added to $\mathcal{F}$, until either there are no more states to add, and the offloading is aborted (lines 21-23), or the algorithm proceeds with the offloading decision. Namely, if the current state doesn't belong to $\mathcal{F}$, no offloading happens at the current t (lines 24-26); otherwise, the offloading of $Br(X_t)$ occurs, if the expected offloading energy is still less than that using local execution (lines 27-39).
- **Water-Filling with Exponentiation (WF-Exp) (Algorithm 3)**: This variation uses exponentiation of the MC transition matrix in order to calculate the expected finishing time and the number of offloading time slots to offload the remaining job. Obviously, this is a more accurate approximation than the one performed by Algorithm 2, since it takes into account the current state and the exact number of steps needed in order to reach another state. More specifically, Algorithm 3 replaces lines 8-10, 16-18, 27-29 of Algorithm 2 with lines 14-16, 21-23, 32, respectively.

B. Water-Filling with Scheduling

This algorithm is a more sophisticated version of the previous Water-Filling algorithm. Again, there are two variations, one that uses equilibrium probabilities for its computation, and one that uses transition probabilities matrix exponentiation.

- **Water-Filling with Equilibrium and Scheduling (WF-Equ-Sch) (Algorithm 4)**: The algorithm works similarly

Algorithm 2 Water-Filling with Equilibrium (WF-Equ)

```
Input: Local execution starting time $t_L$, local execution energy $E_L$, job deadline $t_D$, equilibrium probabilities $\pi$, state space of the Markov chain $\mathcal{M} = \{1, 2 \ldots, m\}$, current state $X_t$, remaining size $S_r = S$.
 1: for all $t = 1, \ldots, t_D$ do
 2:     if $S_r = 0$ then
 3:         break
 4:     end if
 5:     $\mathcal{F} = \{m\}$                ▷ $m$ is the highest bit-rate state
 6:     $B_{avg} = \sum_{i \in \mathcal{F}} \pi[i] Br[i]$
 7:     if $X_t \notin \mathcal{F}$ then
 8:         $t_f = t + \frac{S_r}{B_{avg}} + T_{rest}$
 9:     else
10:         $t_f = t + \frac{S_r - Br(X_t)}{B_{avg}} + T_{rest}$
11:     end if
12:     while $(t_f > t_D)$ $and$ $(\mathcal{M} \neq \mathcal{F})$ do
13:         add the state with the highest bit rate from the set $\mathcal{M} - \mathcal{F}$ to $\mathcal{F}$
14:         $B_{avg} = \sum_{i \in \mathcal{F}} \pi[i] Br[i]$
15:         if $X_t \notin \mathcal{F}$ then
16:             $t_f = t + \frac{S_r}{B_{avg}} + T_{rest}$
17:         else
18:             $t_f = t + \frac{S_r - Br(X_t)}{B_{avg}} + T_{rest}$
19:         end if
20:     end while
21:     if $t_f > t_D$ then
22:         break
23:     end if
24:     if $X_t \notin \mathcal{F}$ then
25:         continue                         ▷ Move to time $t + 1$
26:     end if
27:     $B_F = \frac{\sum_{i \in \mathcal{F}} \pi[i] Br[i]}{\sum_{i \in \mathcal{F}} \pi[i]}$
28:     $E_{up} = (1 + \frac{\max\{0, S_r - Br(X_t)\}}{B_F}) E_{tr}$
29:     $t_f = t + \frac{\max\{0, S_r - Br(X_t)\}}{B_{avg}} + T_{rest}$
30:     $E_l = (\max(t_f + 1, t_L) - \max(t, t_L)) E_L / T_L$
31:     if $t_f \leq t_D$ then
32:         $E_{off} = E_{up} + E_l + T_{down} E_{rc}$
33:     else
34:         $E_{off} = \frac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$
35:     end if
36:     if $E_{off} < \frac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$ then
37:         Offload at $t$
38:         $S_r = \max\{0, S_r - Br(X_t)\}$
39:     end if
40: end for
```

Algorithm 3 Water-Filling with Exponentiation (WF-Exp)

```
Input: Local execution starting time $t_L$, local execution energy $E_L$, job deadline $t_D$, transition probability matrix $P$, state space of the Markov chain $\mathcal{M} = \{1, 2 \ldots, m\}$, current state $X_t$, remaining size $S_r = S$.
 1: for all $t = 1, \ldots, t_D$ do
 2:     if $S_r = 0$ then
 3:         break
 4:     end if
 5:     for all $i = 1, \ldots, m$ do
 6:         if $i == X_t$ then
 7:             $\alpha[i] = 1$
 8:         else
 9:             $\alpha[i] = 0$
10:         end if
11:     end for
12:     $\mathcal{F} = \{m\}$                ▷ $m$ is the highest bit-rate state
13:     if $X_t \notin \mathcal{F}$ then
14:         Find minimum $t_f$ s.t. $S_r - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
15:     else
16:         Find minimum $t_f$ s.t. $S_r - Br(X_t) - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
17:     end if
18:     while $(t_f > t_D)$ $and$ $(\mathcal{M} \neq \mathcal{F})$ do
19:         add the state with the highest bit rate from the set $\mathcal{M} - \mathcal{F}$ to $\mathcal{F}$
20:         if $X_t \notin \mathcal{F}$ then
21:             Find minimum $t_f$ s.t. $S_r - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
22:         else
23:             Find minimum $t_f$ s.t. $S_r - Br(X_t) - \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot Br[i] \leq 0$
24:         end if
25:     end while
26:     if $t_f > t_D$ then
27:         break
28:     end if
29:     if $X_t \notin \mathcal{F}$ then
30:         continue                         ▷ Move to time $t + 1$
31:     end if
32:     $E_{up} = \sum_{k=t+1}^{t_f} \sum_{i \in \mathcal{F}} \left(\alpha P^{k-t}\right)[i] \cdot E_{tr}$
33:     $E_l = (\max(t_f + 1, t_L) - \max(t, t_L)) E_L / T_L$
34:     if $t_f \leq t_D$ then
35:         $E_{off} = E_{up} + E_l + T_{down} E_{rc}$
36:     else
37:         $E_{off} = \frac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$
38:     end if
39:     if $E_{off} < \frac{t_D - \max\{t, t_L\} + 1}{T_L} E_L$ then
40:         Offload at $t$
41:         $S_r = \max\{0, S_r - Br(X_t)\}$
42:     end if
43: end for
```

to WF-Equ, with the important difference that it tries to figure out the *best* offloading finishing time t_f. In order to do that, the algorithm goes over all possible values for t_f from t to t_D. For each t_f, it applies the calculations of Algorithm 2, but on time range $t_f - t$ (instead of $t_D - t$), in order to calculate the expected offloading energy cost. Note that sets $\mathcal{F}$ may differ for different t_f. Then, the algorithm picks the minimum offloading energy consumption calculated over all finishing times and compares this minimum value to the energy required to process the rest of job locally and offloads at t if the former is smaller than the latter. Algorithm 4 is the equilibrium variation of Water-Filling with Scheduling.

Algorithm 4 Water-Filling with Equilibrium and Scheduling (WF-Equ-Sch)

```
Input: Local execution starting time $t_L$, local execution energy $E_L$, job deadline $t_D$, equilibrium probabilities $\pi$ , total number of states $M$, current state $X_t$, remaining size $S_r$.
1: for all $t = 1, \ldots, t_D$ do
2:     for all $t_f = t, \ldots, t_D$ do
3:         Calculate the expected offloading cost $E_{off}^{(t_f)}$ for period $[t, \ldots, t_f]$ using Algorithm 2
4:     end for
5:     $E_{off} = \min_{t \leq t_f \leq T_D} E_{off}^{(t_f)}$
6:     if $E_{off} < \frac{t_D - \max\{t, t_L\}+1}{T_L} E_L$ then
7:         Offload at $t$
8:     end if
9: end for
```

- **Water-Filling with Exponentiation and Scheduling (WF-Exp-Sch)**: This variation of WF-Equ-Sch uses transition matrix exponentiation instead of equilibrium probabilities in its calculations, exactly as WF-Exp does.

C. Generalized Water-Filling

This algorithm is a generalization of the first Water-Filling algorithm, and its two variations are as follows:

- **Generalized Water-Filling with Equilibrium (Gen-WF-Equ)**: This algorithm is a generalization of WF-Equ. Instead of defining a single set $\mathcal{F}$ (when it exists) as the minimal prefix of the state ordering according to bitrates that allow offloading before the deadline, it considers *all* possible such prefixes, and keeps the best. More specifically, if the states $S_1, S_2, \ldots, S_M$ are ordered in decreasing bitrates, and $\mathcal{F}_1$ as defined by Algorithm 2 is $\mathcal{F}_1 = \{S_1, S_2, \ldots, S_l\}$, then the algorithm repeats the same computations for sets $\mathcal{F}_1 = \{S_1, S_2, \ldots, S_l\}, \mathcal{F}_2 = \{S_1, S_2, \ldots, S_{l+1}\}, \ldots, \mathcal{F}_{M+1-l} = \{S_1, S_2, \ldots, S_M\}$, and keeps the set of minimum expected offloading energy cost; then it proceeds exactly as in WF-Equ.
- **Generalized Water-Filling with Exponentiation (Gen-WF-Exp)**: This algorithm is the same as Gen-WF-Equ, except that it uses transition matrix exponentiation instead of equilibrium probabilities in its calculations.

As will become apparent by the simulation results in Section VI, the use of exponentiation by our heuristics dramatically increases their running times, and, therefore, their use is mostly as benchmarks for their versions that use equilibrium calculations.

VI. Simulation Results

In this section, computer simulation is used to study the performance of the proposed preemptive offloading algorithms. In accordance with concurrent local execution, in all algorithms, local execution starts at time slot t_L if offloading (includes uploading to, remote execution at, and downloading from the server) is not completed at time slot $t_L - 1$. This ensures that the job deadline is satisfied either locally or remotely. In order to investigate the performance of the preemptive offloading algorithms in different wireless transmission conditions, we use four different Markov chains to model the communication channel. Two channels are modelled by a 9-state Markov chain shown in Figure 3 and two are modelled by a 5-state Markov chain shown in Figure 4. In both cases, state 1 represents the channel outage that may occur because of shadowing or other similar phenomena; and for the other states, higher states have higher bit rates. In the 9-state channels, the assigned bit rates to states 1 to 9 are $0, 6, 9, 12, 18, 24, 36, 48, 54$ K bits per time slot, respectively, which are the data rates of the IEEE 802.11g standard. In the 5-state channels, the assigned bit rates to states 1 to 5 are $0, 12.5, 25, 37.5$ and 50 K bits per time slot, respectively. For each of the Markovian channel models, we consider two state transition probability matrices, as given in (15)-(18), where $\mathbb{P}_1$ and $\mathbb{P}_2$ are for the 9-state model, and $\mathbb{P}_3$ and $\mathbb{P}_4$ are for the 5-state model. The channels with $\mathbb{P}_1$ and $\mathbb{P}_3$ have equal steady state probability at all the states and are referred to as "uniform channels", and that with $\mathbb{P}_2$ and $\mathbb{P}_4$ have higher steady state probabilities at the lowest (outage) and the highest states and are referred to as "non-uniform channels". The intent behind the channel model selections is to create channels with varying degrees of channel state predictability, so that differences between the algorithms can be properly assessed. All these channels have approximately the same average channel bit rate.

$$\mathbb{P}_1 = \begin{bmatrix} 0.12 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 & 0.11 \\ 0.10 & 0.55 & 0.35 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.30 & 0.30 & 0.30 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.30 & 0.30 & 0.30 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.30 & 0.30 & 0.30 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.30 & 0.30 & 0.30 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.30 & 0.30 & 0.30 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.30 & 0.30 & 0.30 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.30 & 0.60 \end{bmatrix} \tag{15}$$

$$\mathbb{P}_2 = \begin{bmatrix} 0.84 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 & 0.02 \\ 0.10 & 0.05 & 0.85 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.25 & 0.05 & 0.60 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.25 & 0.05 & 0.60 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.25 & 0.05 & 0.60 & 0.00 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.25 & 0.05 & 0.60 & 0.00 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 0.05 & 0.60 & 0.00 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.25 & 0.05 & 0.60 \\ 0.10 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.05 & 0.85 \end{bmatrix} \tag{16}$$

$$\mathbb{P}_3 = \begin{bmatrix} 0.40 & 0.15 & 0.15 & 0.15 & 0.15 \\ 0.15 & 0.55 & 0.30 & 0.00 & 0.00 \\ 0.15 & 0.30 & 0.25 & 0.30 & 0.00 \\ 0.15 & 0.00 & 0.30 & 0.25 & 0.30 \\ 0.15 & 0.00 & 0.00 & 0.30 & 0.55 \end{bmatrix} \tag{17}$$

$$\mathbb{P}_4 = \begin{bmatrix} 0.88 & 0.03 & 0.03 & 0.03 & 0.03 \\ 0.10 & 0.05 & 0.85 & 0.00 & 0.00 \\ 0.10 & 0.30 & 0.05 & 0.55 & 0.00 \\ 0.10 & 0.00 & 0.30 & 0.05 & 0.55 \\ 0.10 & 0.00 & 0.00 & 0.05 & 0.85 \end{bmatrix} \tag{18}$$

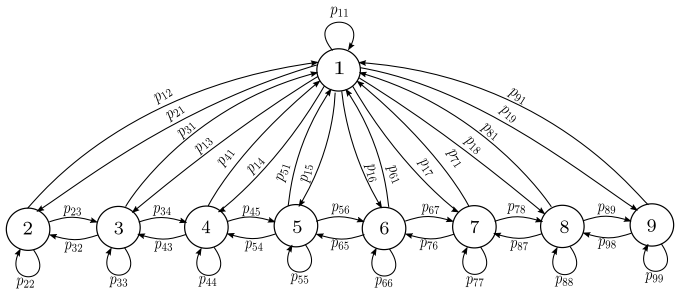


Fig. 3: Markov Chain for $\mathbb{P} = \mathbb{P}_1$ and $\mathbb{P}_2$

The default parameters used in the experiments are given as follows. Each time slot is taken to be 1 msec, which is also used to normalize all system time values. The transmit and receive power is 1 W and 0.5 W, respectively, which corresponds to the transmission and receive energy during each time slot as $E_{tr} = 1$ mJ and $E_{rc} = 0.5$ mJ, respectively. A job with computational load $D = 10$M CPU cycles is considered. The job completion deadline T_D is set to 60 time slots. The local execution energy per CPU cycle is $v_l = 2 \times 10^{-6}$ mJ and the local computation speed is $f_l = 1$M CPU cycles per time slot [20], [21]. Therefore, the local execution time is $T_L = D/f_l = 10$ time slots, and the local energy consumption $E_L = v_l D = 20$ mJ. We consider that the remote execution time is $T_{exec} = 1$ time slot, i.e., the remote processing speed is 10 times that of local processing. The download time T_{down} is assumed to be 1 time slot. In all the simulations, we collect both average energy consumption of the mobile device and the running time of the algorithms, where the running time is the average amount of time needed to make the offloading decision at one time slot. Each value of average energy consumption and running time is obtained after repeating the simulation for 1,000 runs. In addition, for the exponentiation-based algorithms, namely, WF-Exp, Gen-WF-Exp, and WF-Exp-Sch, the calculation of the channel matrix exponentiation is not counted in the running time because this work can be done in the background before the mobile device initiates an offload.

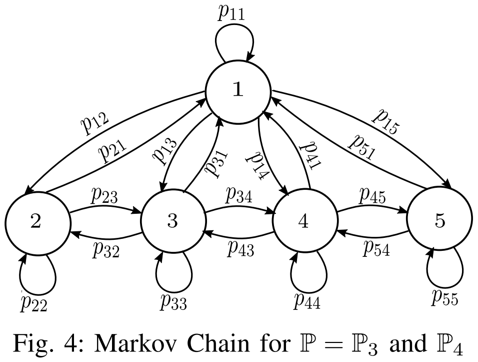


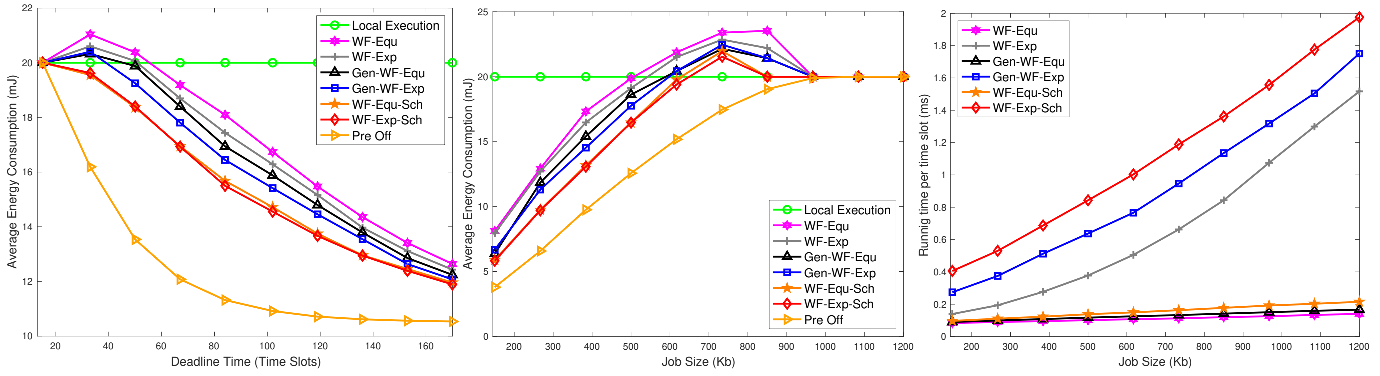
Fig. 4: Markov Chain for $\mathbb{P} = \mathbb{P}_3$ and $\mathbb{P}_4$

For comparison, we also plot the offline bound (Pre Off) given in Section III and *Local Execution* that executes the entire job locally without doing any offloading. When collecting the simulation results, the energy consumption for running the online algorithm at the mobile device was assumed to be negligible compared to that for transmitting to the cloud server. This is a common assumption when the amount of data for uploading the task is large. Despite this, we have included graphs of the relative running times of the algorithms so that this component could be included if required.

Figure 5 shows simulation results of the offloading algorithms over the 9-state wireless channel with state transition probability matrix $\mathbb{P}_1$. Figure 5a shows the average energy consumption of the mobile device as the job deadline T_D changes. When T_D is small, offloading cannot meet the tight delay budget even when the channel is always at the best state. In this case, all the offloading algorithms decide to not offload, resulting in the same energy consumption as Local Execution. As T_D increases, the heuristic algorithms may decide to offload at some time slots but offloading most likely cannot be completed before t_L due to the small time budget. This triggers local execution, and results in overall energy consumption that is higher than Local Execution. For a certain range of T_D, the energy consumption may increase with T_D and then decrease. This is because as T_D increases, the offloading algorithms all attempt to upload at more time slots, while T_D is still insufficient to allow offloading to be completed in time. As T_D further increases, offloading may be completed before T_D, which reduces the concurrent local execution energy consumption; and as T_D further increases, offloading is more likely completed before t_L, in which case, concurrent local execution is not needed, and the average energy consumption of the mobile device using the offloading algorithms further decreases with T_D.

By comparing the different offloading algorithms, we find that in general, using "Exponentiation" (i.e., the *Exp* algorithms) helps reduce the average energy consumption, compared with using "Equilibrium" computations (i.e., the *Equ* algoritms) only. Using the "Generalization" approach helps reduce the average energy consumption; and using "Scheduling" can further reduce the average energy consumption, compared with using "Generalization". Overall, the WF-Equ-Sch and WF-Exp-Sch algorithms achieve the lowest average energy consumption among all the offloading algorithms.

Comparing with the offline bound, the average energy consumption of using the heuristic offloading algorithms is the same as the bound when T_D is very small as all the algorithms decide to not offload. As T_D increases, the gap between the offline bound and the heuristic offloading algorithms increases first, then decreases. When T_D is sufficiently large, the heuris-



(a) S_{up}=500 kb (b) T_D=60 ms (c) T_D=60 ms

Fig. 5: 9-states uniform distribution: $\mathbb{P} = \mathbb{P}_1$

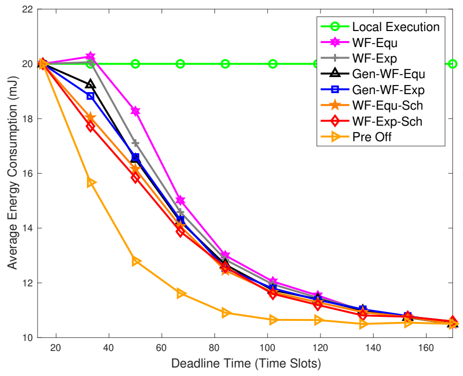
tic offloading algorithms may decide to upload only when the channel is in the best state, and the average energy consumption of the offloading algorithms asymptotically approaches the offline bound.

Figure 5b shows the average energy consumption of the mobile device as the job size S_{up} changes. When S_{up} is small, the offloading algorithms may decide to upload only when the channel condition is sufficiently good, and offloading most likely can be completed before t_L. In this case, the average energy consumption of the offloading algorithms is close to the offline bound. As S_{up} increases, it becomes less likely that the task can be completed before t_L or even before T_D by offloading to the server. In this case, more concurrent local execution is needed that increases the energy consumption of the mobile device. When S_{up} is sufficiently large, all the offloading algorithms decide to not offload, resulting in the same energy consumption as in Local Execution.

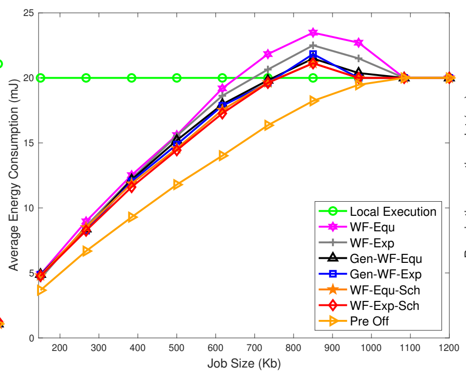
Figure 5c shows the running time versus job size for the offloading algorithms. As can be seen from this figure, the WF-Exp, WF-Exp-Sch and Gen-WF-Exp algorithms are much more time consuming than WF-Equ, WF-Equ-Sch and Gen-WF-Equ. This is mainly due to a more complicated process to find the "most efficient" set $\mathcal{F}$ that considers the number of steps needed from the current channel state to other states in order to finish uploading the task. As a result, the running time of the exponentiation-based algorithms increases quickly with the task size, because more channel transition steps should be checked. In contrast, the equilibrium-based algorithms are much less sensitive to task size increase in terms of running time. Although the longer running time of the exponentiation-based algorithms makes them less practical in online situations, they do make more accurate offloading decisions than the equilibrium-based algorithms. Meanwhile we also notice that, the equilibrium-based algorithms (e.g., WF-Equ-Sch) achieve the average energy consumption very much close to the corresponding exponentiation-based algorithms (e.g., WF-Exp-Sch). Furthermore, doing generalization and scheduling increases the running time, and the running time of WF-Equ-Sch is slightly larger than that of Gen-WF-Equ, which is slightly larger than that of WF-Equ, although the running time of all the three equilibrium-based algorithms is very much close to each other.

Figure 6 shows simulation results of the offloading algorithms over the 9-state wireless channel with state transition probability matrix $\mathbb{P}_2$, which represents a non-uniform channel. Comparing Figures 6a and 5a we find that the non-uniform channel results in lower average energy consumption than the uniform channel when T_D is small; and as T_D increases, the curves in Figure 6a drop and approach the offline bound much faster than in Figure 5a. The non-uniform property of the channel helps the online algorithms make better offloading decisions and save energy consumption, compared to the uniform channel. For this reason, the performance difference between different offloading algorithms is much smaller than in the uniform-channel case. Similar observations can be obtained by comparing Figures 6b and 5b, from which we can see that the curves in Figure 6b rise slower with S_{up} than in Figure 5b. All the offloading algorithms have almost the same energy consumption performance except for a small range of S_{up} where the average energy consumption is above the Local Execution energy. Figure 6c further shows that the running time of WF-Exp, WF-Exp-Sch and Gen-WF-Exp is much more than WF-Equ, WF-Equ-Sch and Gen-WF-Equ, which is consistent with Figure 5c, and the non-uniform property of the channel does not affect the running time of the algorithms in an obvious way.

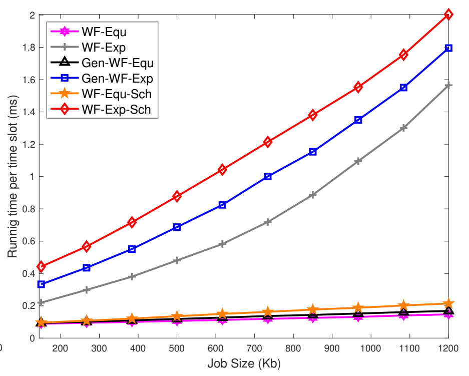
Figure 7 shows simulation results of the offloading algorithms over the 5-state wireless channel with state transition probability matrix $\mathbb{P}_3$, which represents a uniform channel. Comparing Figures 7a and 5a we find that the 5-state channel results in approximately the same average energy consumption as the 9-state uniform channel when T_D is small; and as T_D increases, the curves in Figure 7a drop and approach the offline bound much faster than in Figure 5a. The smaller number of channel states makes it easier for the online algorithms to make better offloading decisions that helps improve the energy consumption performance. Similar observations can be obtained by comparing Figures 7b and 5b, which shows that the average energy in Figure 7b increases slightly faster with S_{up} than in Figure 5b. Figure 7c further shows that the running time of the equilibrium-based algorithms (i.e., WF-Equ, WF-Equ-Sch and Gen-WF-Equ) is much less than that



(a) S_{up}=500 kb

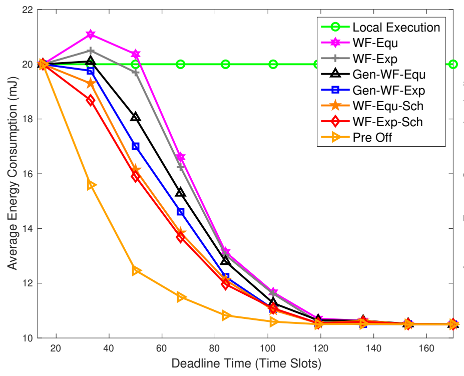


(b) T_D=60 ms

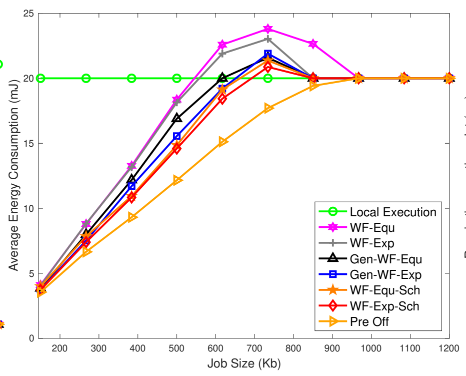


(c) T_D=60 ms

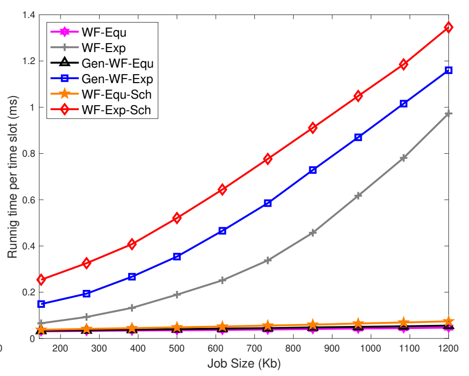
Fig. 6: 9-states non-uniform distribution: $\mathbb{P} = \mathbb{P}_2$



(a) S_{up}=500 kb



(b) T_D=60 ms



(c) T_D=60 ms

Fig. 7: 5-states uniform distribution: $\mathbb{P} = \mathbb{P}_3$

of the exponentiation-based ones (i.e., WF-Exp, WF-Exp-Sch and Gen-WF-Exp) and not much affected by the task size increase. In addition, by comparing Figures 5c and 7c, we find that the running time of the algorithms in the 9-state uniform channel is higher than that in the 5-state channel case for all the algorithms, since more calculations are needed for the channel with more states.

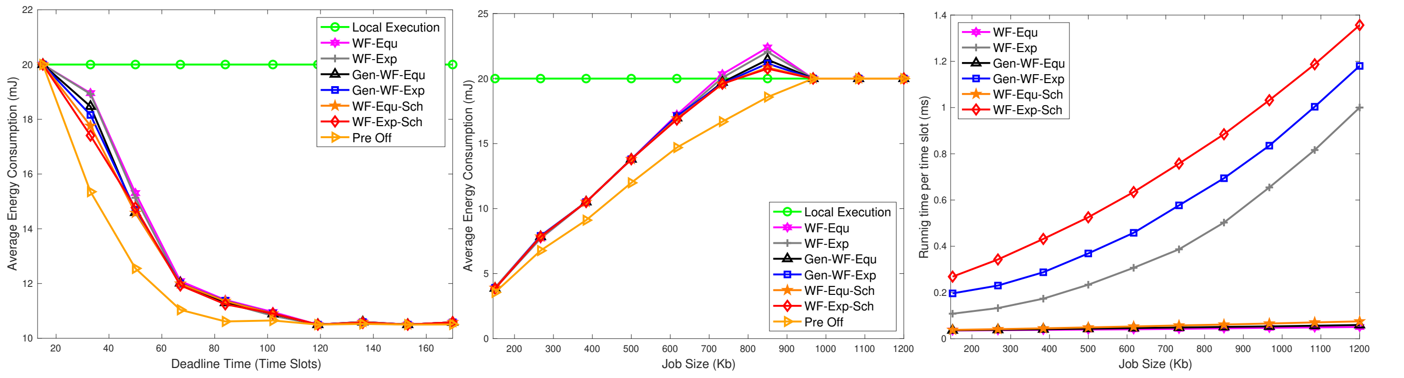
Figure 8 shows simulation results of the offloading algorithms over the 5-state wireless channel with state transition probability matrix $\mathbb{P}_4$, which represents a non-uniform channel. Comparing with Figure 7, in terms of energy consumption, there is only a slight difference between the different offloading algorithms, and the performance of these heuristic offloading algorithms is very close to the offline optimum. The running time of the algorithms in this case is not obviously different from that in the 5-state uniform channel case.

In summary, among all the heuristic algorithms, WF-Exp-Sch achieves the lowest mobile device energy consumption and WF-Equ consumes the shortest running time, while WF-Equ-Sch is the best choice because its energy consumption is very much close to WF-Exp-Sch and its running time is almost the same as WF-Equ. The difference between the heuristic algorithms is relatively small for very large size tasks with tight completion time or very small size tasks with loose completion time, because the decision is most likely to always execute the task locally (for the former) or always offload (for the latter). In terms of running time (complexity), the exponentiation-based algorithms are sensitive to the number of channel states and the task size, while the equilibrium-based algorithms are not as much affected by these parameters. In terms of energy consumption of the mobile device, the difference among the heuristic algorithms is more obvious in the uniform channel than in the non-uniform channel case.

All the above results are generated based on the parameter setting that has the cloud server CPU processing speed 10 times of the local CPU speed. Given the local processing speed, if the processing speed at the cloud server is higher, the probability that offloading can meet the delay constraint of the task is higher, and more energy consumption of the mobile device may be saved by offloading. However, this benefit is eventually limited by the quality of the wireless channel, i.e., the amount of time and energy needed for uploading/downloading the task.

VII. Conclusions

This paper has studied preemptive mobile computation offloading, when concurrent local execution (CLE) is used to guarantee task execution time constraints. In CLE, local task execution is permitted even if a remote offload decision has been initiated. This mechanism ensures that hard task



(a) S_{up}=500 kb

(b) T_D=60 ms

(c) T_D=60 ms

Fig. 8: 5-states non-uniform distribution: $\mathbb{P} = \mathbb{P}_4$

deadlines are always satisfied. In preemptive offloading, a decision is made to either continue offloading or to temporarily interrupt the offload at the start of each time slot. This gives the system the ability to adapt to changes in wireless channel conditions during the offload. The homogeneous Markovian wireless channel case was considered. An online computation offloading algorithm, referred to as Optimal Preemptive Offloading (OPO), was formulated for preemptive offloading, and was shown to be energy-optimal. The computational complexity of OPO is prohibitive, even for simple Markovian channels, and, therefore, the paper introduced three computationally efficient techniques: *Water-Filling*, *Water-Filling with Scheduling*, and *Generalized Water-Filling*. For each, two variations were considered. The first (*Equ*) uses the equilibrium channel state probabilities to determine its offloading decisions, and the second (*Exp*) uses Markovian transition matrix exponentiation. The six resulting algorithms have a wide variety of energy performance and computational complexity. The performance of the proposed algorithms was compared on Markovian channels with different characteristics, in order to show the tradeoffs between complexity and mobile energy saving performance.

Future directions: There are several possible directions for future extensions of our work. The high computational complexity incurred by the exact DP solutions lead us to the development of approximate heuristics; rollout techniques (cf. [22], and the references therein) can be studied as an efficient mechanism to tradeoff complexity and optimality. Reinforcement learning, which has already been used for designing offloading policies (e.g., [23], [24] can be used to effectively learn parameters of the system (e.g., channel transition probabilities).[1]

References

[1] H. Ba, W. Heinzelman, C.-A. Janssen, and J. Shi, "Mobile computing - A green computing resource," in *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, July 2013, pp. 4451–4456.

[2] G. Huerta-Canepa and D. Lee, "A virtual cloud computing provider for mobile devices," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing Services: Social Networks and Beyond*, June 2010, p. 6.

[1]We are grateful to an anonymous reviewer for pointing out these references to us.

[3] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "CloneCloud: Elastic Execution between Mobile Device and Cloud," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966473

[4] B.-G. Chun and P. Maniatis, "Augmented Smartphone Applications Through Clone Cloud Execution," in *Proceedings of 12th Conference Hot Topics Operating Systems*, 2009, p. 8.

[5] M. Satyanarayanan, P. Bahl, and R. Cáceres, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct.—Dec. 2009.

[6] K. Kumar and Y.-H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" *IEEE Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[7] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, March 2012, pp. 2716—2720.

[8] O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Optimization of Radio and Computational Resources for Energy Efficiency in Latency-Constrained Application Offloading," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4738–4755, October 2015.

[9] K. Kumar and Y.-H. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[10] H. A. Lagar-Cavilla, N. Tolia, E. De Lara, M. Satyanarayanan, and D. OHallaron, "Interactive resource-intensive applications made easy," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2007, pp. 143–163.

[11] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 14, no. 1, pp. 81–93, 2014.

[12] Y. Liu and M. J. Lee, "An effective dynamic programming offloading algorithm in mobile cloud computing system," in *Proceedings of IEEE International Conference on Wireless Communications and Networking (WCNC)*, April 2014, pp. 1868–1873.

[13] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications*, vol. 12, no. 9, pp. 4569–4581, 2013.

[14] A. Hekmati, P. Teymoori, T. D. Todd, D. Zhao, and G. Karakostas, "Optimal mobile computation offloading with hard deadline constraints," *IEEE Transactions on Mobile Computing*, 2019.

[15] ——, "Optimal multi-decision mobile computation offloading with hard task deadlines," in *IEEE Symposium on Computers and Communications (ISCC)*, 2019.

[16] ——, "Optimal multi-part mobile computation offloading with hard deadline constraints," *Comput. Commun.*, vol. 160, pp. 614–622, 2020.

[17] X. Chen, "Decentralized Computation Offloading Game for Mobile Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 974–983, April 2015.

[18] H. Cao and J. Cai, "Distributed Multiuser Computation Offloading for Cloudlet-Based Mobile Cloud Computing: A Game-Theoretic Ma-

chine Learning Approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 752–764, January 2018.

[19] G. Peskir and A. Shiryaev, *Optimal stopping and free-boundary problems*, ser. Lectures in Mathematics ETH Zurich. Dordrecht: Springer, 2006.

[20] M. Nir, A. Matrawy, and M. St-Hilaire, "An energy optimizing scheduler for mobile cloud computing environments," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE, 2014, pp. 404–409.

[21] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *Wireless Communications, IEEE Transactions on*, vol. 11, no. 6, pp. 1991–1995, 2012.

[22] D. P. Bertsekas, "Dynamic programming and suboptimal control: A survey from ADP to MPC," *Eur. J. Control*, vol. 11, no. 4-5, pp. 310–334, 2005.

[23] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks, HotNets 2016*, 2016, pp. 50–56.

[24] 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*, 2018, pp. 1–6.