Optimal Multi-Part Mobile Computation Offloading With Hard Deadline Constraints*

Arvin Hekmati[a], Peyvand Teymoori[b], Terence D. Todd[b], Dongmei Zhao[b], George Karakostas[c,*]

[a]*Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, California, USA*
[b]*Department of Electrical and Computer Engineering, McMaster University, Hamilton, Ontario, Canada*
[c]*Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada*

Abstract

This paper considers mobile computation offloading when concurrent local execution (CLE) is used to enforce task execution time constraints. This mechanism can ensure that hard task deadlines are satisfied regardless of any randomness induced by the wireless channel, network or cloud servers. In this type of system, mobile device energy may be reduced by segmenting a task upload into multiple parts, rather than doing a conventional contiguous task upload. The paper uses this mechanism to adapt to changes in channel conditions during the offload. Unlike the contiguous task offload case however, the upload initiation times of each part must be determined dynamically. This is done while ensuring that hard task deadlines are always satisfied. In this paper, the multi-part computation offloading case is considered. In multi-part offloading, the task to be offloaded is partitioned into K upload parts before any offload initiation decisions are made. In this case, current channel state information is incorporated into the offload decisions, and the system must always satisfy a hard task execution time constraint using concurrent local execution. The paper considers the case for Markovian wireless channels. A provably energy-optimal online computation offloading algorithm (MuliOpt) is introduced for multi-part offloading. MultiOpt is shown to be optimal using Markovian decision process stopping theory. Since the computational complexity of MultiOpt can be significant, simpler and more computationally efficient heuristics, which also respect the hard task execution deadline, may be used. The paper introduces two such heuristics, the Immediate Offloading, and Multi Threshold algorithms. The mobile energy use of MultiOpt is compared to these heuristics, as well as to local execution without offloading and an offline energy bound. Simulation results show that MultiOpt performs significantly better when compared to the proposed heuristics, as well as when K increases.

Keywords: Cloud computing, mobile computation offloading, energy efficiency, mobile task execution performance, hard job deadline constraints

1. Introduction

Mobile device energy consumption can sometimes be improved by using mobile computation offloading (MCO). This is accomplished by having the mobile device arrange for tasks to be executed remotely, rather than running them locally on the device itself. Remote execution is typically done on infrastructure-based cloud servers that are accessed by the mobile device using wireless communication channels. There is a large prior literature that deals with the various issues around performing computation offloading [1] [2] [3] [4] [5].

A wide variety of systems have been proposed for mobile computation offload implementation. References [6] and [3] have proposed architectures that control the offloading using facilities from the Microsoft .NET and Android-based APIs, respectively. Computation offloading typically reduces execution energy consumption but usually incurs communication energy costs since the tasks to be executed (and their inputs/outputs) must be transferred over wireless network links. This may also incur additional latency that would not occur otherwise. The tradeoff between energy saving and computing performance for single device offloading has been studied extensively [7] [8] [9].

In the multi-user case, References [10] [11] [12] [13] have considered single task offloading where the entire task is either offloaded or executed locally. Various studies have also considered the case where a given job may be partitioned into multiple tasks, each of which may be executed locally or offloaded [3] [6] [14] [15] [16]. This work raises the issues of how to perform the partitioning, and which tasks should be offloaded given that there may be data dependencies between the tasks. There is also work that models MCO as a competitive game. In this type of system there may be resource contention among the devices, such as the case where they share communication channels and/or cloud servers [17] [18] [19] [20].

Early mobile computation offloading work tended to take a static view of the communication channels used during the offload. More recently, work has considered the case where the mobile device must interact with cloud server(s) over stochastic transmission channels and/or variable network conditions. As an example, Reference [7] models the communication energy using statistical inputs, but the wireless channel is assumed to

*Research supported by NSERC Discovery Grants
*Corresponding author
Email addresses: hekmati@usc.edu (Arvin Hekmati), teymoorp@mcmaster.ca (Peyvand Teymoori), todd@mcmaster.ca (Terence D. Todd), dzhao@mcmaster.ca (Dongmei Zhao), karakos@mcmaster.ca (George Karakostas)

be static throughout the offload, i.e., the channel is quasi-static. In this work, random network conditions are addressed using prediction.

The importance of task execution time constraints for interactive applications was highlighted in Reference [21]. This paper also discussed problems of achieving this, when offloading using stochastic wireless channels. Energy optimal cloud computing was addressed in Reference [22], which included a treatment of task execution time deadlines assuming random wireless channels. Although task completion time deadlines were addressed, their observance was considered statistically rather than as a hard constraint which has to be satisfied. Dynamic programming was used to optimize computation offloading decisions in [23], but task execution time deadlines were not considered. CPU frequency scheduling was used in [24] to control mobile task execution time, when remote offloading occurs under random wireless channel conditions. This was combined with mobile transmit power control, to target task deadlines. However, since there are limits on radio transmit power and CPU frequency, satisfying execution time constraints is not always possible. For this reason, a violation parameter was used to characterize execution time deadline misbehaviour. This reference assumed the well-known Gilbert-Elliot Markovian channel model, as is the case in our simulation results.

Reference [25] used concurrent local execution (CLE), in order to ensure the satisfaction of hard deadlines. In CLE, a hard deadline is guaranteed by permitting simultaneous remote and local task execution when it is needed to ensure task completion times (cf. [25] [26]). Reference [25] proposed an on-line mobile computation offloading algorithm, OnOPT, that was shown to be energy optimal and satisfies hard deadline constraints. Reference [26] 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 uploading data into two parts.

Our paper generalizes [26], by extending multi-part mobile computation offloading to an arbitrary number of parts (instead of just two) when CLE is used. More specifically, the task to be remotely executed is segmented into K parts, with K associated upload initiation time decisions. We assume that both K, as well as the bit-sizes of the K parts are predetermined. Multi-part offloading can be used to reduce mobile energy use when wireless channel conditions change during the computation offload. In multi-part offloading, at the end of uploading one part, the wireless channel may have deteriorated, and therefore a decision to wait until more favourable channel conditions may lead to lower overall mobile device energy consumption. These decisions have to be made using CLE, so that the system always satisfies the given hard task execution time constraint. Since the task is uploaded in separate parts, separate offload initiation time decisions are needed for each, so that mobile device energy consumption is minimized.

The paper considers the Markovian wireless channel case. Under this assumption, a new computation offloading algorithm (MultiOpt) is introduced for multi-part offloading. MultiOpt is shown to be energy optimal, in the sense that no other CLE online computation offloading algorithm can achieve a lower mean mobile device energy consumption. This is shown by creating a new Markov process, which incorporates time in the given Markovian channel model, and then, by using optimal Markovian stopping.

The energy performance of MultiOpt is compared to simpler CLE heuristics that also ensure that hard task execution deadlines are observed, namely, Immediate Offloading, and Channel Threshold, as well as to local execution without offloading. Results show that the proposed algorithm can significantly improve mobile device energy consumption compared to the other approaches, while guaranteeing hard task execution deadlines.

2. System Model

We consider the execution of 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. CLE is employed in order to ensure the completion of jobs before their hard deadlines. Our discussion will focus on single job offloads, but each job could be a sub-task associated with multiple local/remote job execution components (e.g., [27] [28]). We assume that each job can be split into K parts for offloading, each with a (known) number of bits to be transmitted through the uplink channel. Splitting the upload in this way can be advantageous when channel conditions change during the offload. For example, it may be better, energy-wise, to delay further uploading when channel conditions worsen, hoping that it will improve in time to complete the computation offload. We refer to this scenario as *K-Part offloading* throughout this paper.

Note that time is taken to be discrete, i.e., quantized into equal length *time slots* that are referred to by their time slot indices. The time slot duration is defined so as to accommodate the channel propagation model discussed in Section 4 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.

S_{up_i}**:** Bit-size of the i^{th} piece, where $S_{up} = S_{up_1} + S_{up_2} + \ldots + S_{up_K}$.

S_{down}**:** Number of bits transmitted through the downlink channel when downloading job results from the cloud.

2.1. Local Execution

Using the model described in [7], i.e., the local execution energy consumption for a job is determined by its CPU workload, we assume that the *local execution energy consumption* E_L, and the *amount of time to complete local execution* T_L, are known at the time of task generation.

We must ensure that the job deadline constraint is always satisfied. Therefore, local execution must start T_L time slots prior to the job deadline, if remote execution results have not arrived by then (cf. Figure 1). That is, the local execution start time is given by $t_L = t_D - T_L + 1$.

2.2. Remote Execution

The K-Part offloading timing sequence is shown in Figure 1. The first job piece begins uploading at t_{o_1}, and ends at t_{f_1}, then T_{W_1} time slots $t^*_{o_k}$ before the second piece begins uploading at t_{o_2}, and so on, until the Kth piece is uploaded. Note that, by definition, if $t_{o_1} > t_D$, then there is only local execution. It is assumed that the uplink channel uses bit rate adaptation to accommodate random variations in channel conditions. As a result, time intervals $T_{up_i} = t_{f_i} - t_{o_i} + 1,\ i = 1, \ldots, K$, are random variables, dependent on the evolution of the uplink channel state as a given upload occurs. After its uploading is complete, the job is executed on the server in T_{exec} time slots, and its execution results are downloaded to the mobile device in T_{down} time slots. We assume that the execution time T_{exec} is assigned when the job is released, 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). We assume that T_{down} is also communicated to the mobile device at this time; more generally, we can treat the dowloaded results as the $K+1$'th job piece transmitted over the channel, but we will avoid doing this, in order to simplify our presentation. In this paper, we assume that power control is used on the downlink, so that T_{down} is known before the upload. Therefore, the total offloading time T_{off} is given by

$$T_{off} = \sum_{i=1}^{K} T_{up_i} + \sum_{i=1}^{K-1} T_{W_i} + T_{exec} + T_{down}, \qquad (1)$$

where T_{W_i} is the number of time slots that elapse after uploading the ith piece and before uploading the $(i+1)$th piece. If the dowloaded results are received before deadline t_D, any local execution of the job is automatically terminated.

In what follows, we define

$$T_{rest} = T_{exec} + T_{down}. \qquad (2)$$

As in many of the references, we assume that the current state of the channel can be determined prior to making the decision to start an offload. 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.

3. Offline Bound

In this section, an offline lower bound on mobile device energy is derived. This bound is used in Section 6 for performance comparisons with various online computation offloading algorithms. Since the bound is offline, we assume that the wireless channel states are known for all future time slots, i.e., we know the bit rate (in bits per time slot) at all times $1 \le t \le t_D$ (recall that t_R is taken to be 1). When a job is released, the bound chooses the job offload times so that its deadline is met and the energy needed is minimized. Let $t_{f_i}(t_{o_i}),\ 1 \le i \le K$, be the upload finishing time as a function of the uploading starting time t_{o_i} for the ith part, and define $t_{f_0}(t_{o_0}) = 0$. E_{tr} and E_{rc} are the energy costs per time slot for channel transmitting and receiving, respectively. The optimal values for t_{o_i} are found by solving integer programming (IP) program (3)-(5).

The first term in (3) is the local execution energy cost incurred, the second term accounts for the energy cost of uploading the K job parts, and the last term is the cost for downloading the results from the cloud. Constraint (4) ensures that the energy used in offloading does not exceed that of executing the job locally. Note that if the IP is infeasible, then there are no feasible uploading start times t_{o_i}, i.e., it is best to execute only locally without offloading.

4. Markovian Channel and the Time-Dilated Absorbing Markov Model

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 $\mathbb{P} = [P_{ij}]$, where P_{ij} 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. Therefore, we will consider the tree-like Markov chain produced by following the evolution of the channel, starting from an initial state at time $t = 1$, and branching out from each state according to the transition probabilities of the CSMC. We refer to this new Markov chain as the corresponding *time-dilated absorbing Markov chain (TDAMC)*. We will denote by X_t a state in this Markov chain, reached after running the channel for t time slots. We will consider subtrees of this TDAMC (such at $TDAMC_1$ below), endowed with energy costs and absorbing states.

The part of the TDAMC that models the offloading progress if the uploading of S_{up_1} is initiated at a time slot t_s, will be denoted as $TDAMC_1$. An example of $TDAMC_1$ is shown in Figure 2. To simplify the exposition, the diagram shows the two-state Gilbert-Elliot channel case, but the procedure is valid for any Markovian channel. In the Gilbert-Elliot case, the channel is modelled by a CSMC with two states $\{G, B\}$ (i.e., a "Good" one with the higher bit rate, and a "Bad" one, respectively), and with transition probabilities $P_{GG}, P_{GB}, P_{BG}, P_{BB}$.

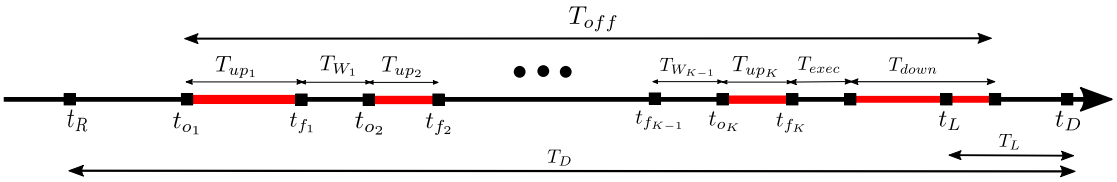

Figure 1: Job offloading timing parameters.

$$\min_{t_{o_1},\ldots,t_{o_K}} \quad \frac{\max(t_{f_K}(t_{o_K})+T_{rest},t_L)-t_L}{T_L}E_L + E_{tr}\sum_{i=1}^{K}(t_{f_i}(t_{o_i})-t_{o_i}+1) + E_{rc}T_{down} \tag{3}$$

$$s.t. \quad \frac{\max(t_{f_K}(t_{o_K})+T_{rest},t_L)-t_L}{T_L}E_L + E_{tr}\sum_{i=1}^{K}(t_{f_i}(t_{o_i})-t_{o_i}+1) + E_{rc}T_{down} \leq E_L \tag{4}$$

$$t_{f_{i-1}}(t_{o_{i-1}})+1 \leq t_{o_i} \leq t_D, \quad i=1,\ldots,K. \tag{5}$$

In each time slot, $TDAMC_1$ transitions to a new state in accordance with these transition probabilities. For clarity, each state s_t^a in the figure is subscripted by its time slot t, and superscripted by a unique identifier a that distinguishes it from the other channel states reachable after t time slots. Hence, the $TDAMC_1$ of Figure 2 models the offloading process initiated at time slot t_s, when the channel state that has been reached at that time is $s_{t_s}^{19}$. The bit rate at each state is also indicated.

In general, $TDAMC_1$ is a rooted subtree of the TDAMC, constructed as follows: The root state is the (known) channel state X_{t_s} at current time slot t_s. At each subsequent time slot, the Markov chain tree branches forward, according to the transitions possible from the current state (X_{t_s}, initially) to other TDAMC states. At each state, the number of job bits transmitted is determined by the bit rate associated with that state. The branching continues to create all possible paths of states needed to upload S_{up_1} bits, up to some state $X_{t_{f_1}}$ corresponding to upload finishing time t_{f_1} for each path from the root (such as $s_{t_s+1}^{37}, s_{t_s+2}^{75}$, represented by squares in Figure 2). At time $t_{f_1}+1$, the second part S_{up_2} is released. Continuing the branching of the TDAMC, and after a possible waiting period, the uploading of S_{up_2} commences, followed by the rest of the K pieces, and the job execution in the cloud in time T_{exec}, and the downloading of the results in time T_{down}, ending in an absorbing state (this part of the offloading for $K=2$ is depicted in Figure 2 as subtrees hanging from states $s_{t_s+2}^{73}, s_{t_s+2}^{74}, s_{t_s+3}^{149}, s_{t_s+3}^{150}$). The optimal waiting time for each path, i.e., the waiting times which optimize the total (over all paths) expected energy cost for uploading $S_{up_2},\ldots,S_{up_K}$, is solved in Section 5. Then the energy cost of each subtree is the optimal expected cost (over all paths) of completing offloading, when uploading S_{up_1} finishes in time slot t_{f_1} and state $X_{t_{f_1}}$. In fact, $TDAMC_1$ does not need to extend all the way into these subtrees, but can treat states $X_{t_{f_1}+1}$ as absorbing states, each with cost equal to the energy cost of its own subtree. This process can obviously be repeated, in order to build the corresponding $TDAMC_i$ for any piece i.

Similarly to [25], the probability of uploading S_{up_i} bits in T_{up_i} time slots, starting at time slot t_{o_i}, and a state $X_{t_{o_i}}$, for $i=1,\ldots,K$, can be calculated by building a separate $TDAMC_i$, with a set of absorbing states $\mathcal{A}_i$, and a set of transient states $\mathcal{T}_i$, for $i=1,\ldots,K$. It encodes the evolution of the channel starting at time slot t_{o_i} and state $X_{t_{o_i}}$, and until S_{up_i} bits are uploaded, at which point an absorbing state in $\mathcal{A}_i$ is reached. Its transition matrix can be written [29] as

$$\mathbb{P}_i = \begin{bmatrix} Q_i & R_i \\ \mathbf{0} & I_{\mathcal{A}i} \end{bmatrix}, \tag{6}$$

where the $|\mathcal{T}_i|\times|\mathcal{T}_i|$ sub-matrix Q_i contains the probabilities of transitioning between transient states, the $|\mathcal{T}_i|\times|\mathcal{A}_i|$ sub-matrix R_i contains the probabilities of transitioning from a transient state to an absorbing state, and $I_{\mathcal{A}i}$ is an $|\mathcal{A}_i|\times|\mathcal{A}_i|$ identity matrix.

The theory of absorbing Markov chains implies that various statistics can be computed by forming the fundamental matrix $N_i=(I-Q_i)^{-1}$, where $N_i[l,m]$ gives the expected number of times that $TDAMC_i$ is in transient state m if the system is started in transient state l. Given the structure of $TDAMC_i$, N_i can be easily decomposed and calculated as in [25], since the particular structure of matrices Q_i, N_i, N_i^{-1} is the same simple one as in [25]. The absorption probabilities matrix W_i for all absorbing states is given by

$$W_i = N_i R_i, \tag{7}$$

where W_i is a $|\mathcal{T}_i|\times|\mathcal{A}_i|$ matrix, and $W_i[l,m]$ gives the probability that absorbing state m will be reached when starting in transient state l. Therefore, the probability of uploading the ith part with size S_{up_i} in T_{up_i} time slots, starting at time slot t_{o_i} and state $X_{t_{o_i}}$, is

$$P_{t_{o_i}}(S_{up_i},T_{up_i},X_{t_{o_i}}) = \textstyle\sum_{j\in\mathcal{S}_i} W_i[X_{t_{o_i}},j], \tag{8}$$

where $\mathcal{S}_i$ is the set of absorbing states in $TDAMC_i$ reached by a path of length $T_{up_i}+1$ from the root $X_{t_{o_i}}$.

For $i=1,2,\ldots,K-1$, if the uploading of S_{up_i} starts at

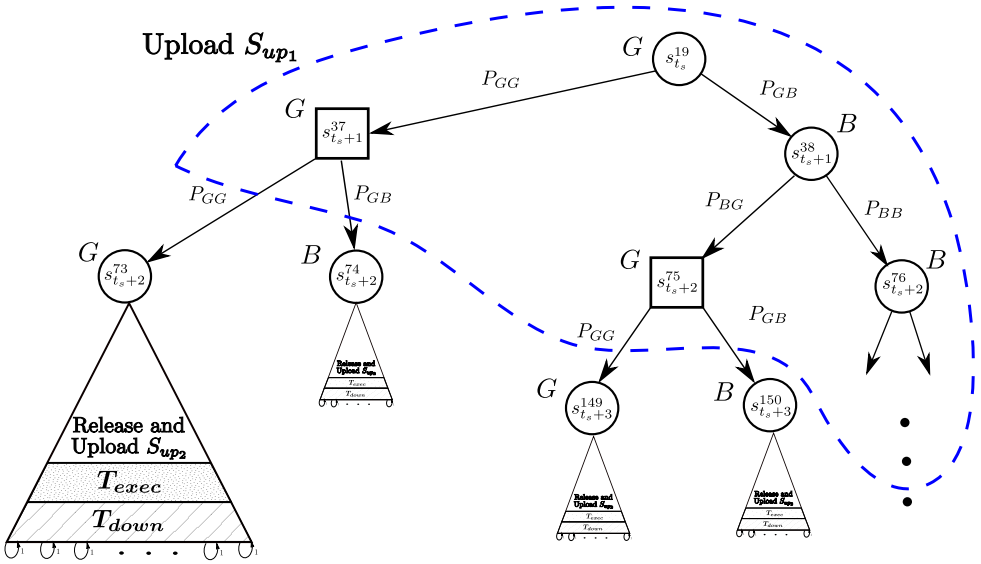

Figure 2: $TDAMC_1$ when offloading S_{up_1} starts at time t_s.

time slot t_{o_i}, the expected offloading energy cost when offloading starts at time slot t_{o_i} in state $X_{t_{o_i}}$ and finishes exactly in T_{up_i} time slots or at t_D (whichever comes first), is given by equation (9), where E_{tr} is the transmission energy of the mobile device during one time slot, while the expected energy cost of uploading S_{up_K} is given by equation (10), where E_{rc} is the energy consumption of the mobile device during one time slot when receiving from the server. Then, the expected offloading energy cost E_{off_i} for $i = 1, 2, \ldots, K$ is computed by equation (11), where B_{max} and B_{min}, respectively, are the bit rates at the best and worst channel states.

Similarly, the local execution energy cost corresponding to the uploading of the ith job piece, for $i = 1, 2, \ldots, K-1$ is given by (12)[1], while the local execution energy cost due to the Kth job piece and the rest of offloading time T_{rest} is given by (13). Then, the expected local execution energy cost E_{L_i} for $i = 1, 2, \ldots, K$ is computed by equation (14).

5. Optimal Algorithm for K-Part Offloading

In this section we use the TDAMC's constructed in Section 4, and the theory of optimal stopping for Markov decision processes [30], in order to define optimal offloading algorithms, and prove their optimality. Recall that, for simplicity, we have set $t_R = 1$. A high-level description of algorithm MultiOpt (cf. Algorithm 1) is as follows: Starting from time slot $t = 1$ (the release time of the job), at each time slot t the algorithm considers $TDAMC_1$ in order to determine the expected cost of the whole offloading process if uploading S_{up_1} commences at the current time t. If that cost is less than the expected offloading cost when the algorithm waits one more time slot, then $t^*_{o_1} = t$ (offloading S_{up_1} commences), otherwise the algorithm postpones its decision for time slot $t+1$. Once the uploading of S_{up_1} finishes, for the rest of the parts the algorithm repeats the same decision process at every time slot (using $TDAMC_i$ to compute expected costs), to determine the time $t^*_{o_i}$ of starting uploading S_{up_i} for $i = 2, 3, \ldots, K$.

At any time slot t (and state X_t), and given that pieces $1, 2, \ldots, i-1$ have already been uploaded, MultiOpt decides the uploading starting time $t^*_{o_i} \geq t$ for S_{up_i}. Its decisions $t^*_{o_i}$ for $i = 1, 2, \ldots, K$ are optimal iff they are the solutions of the minimization problems (one for each i) (15)-(16), where $\mathcal{S}_i$ is the set of states reachable after running the channel until time slot t_{o_i}, $\hat{\mathcal{S}}_i$ is the set of absorbing states of $TDAMC_i$ rooted at $X_{t_{o_i}}$, and $v_{i+1}(t_{f_i}, X_{t_{f_i}+1})$ is the optimal expected energy cost for the rest of the offloading when S_{up_i} finishes uploading at time t_{f_i}, i.e., the cost of the absorbing state $X_{t_{f_i}+1}$ of $TDAMC_i$. In (15)-(16), $g_i(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}})$ is the expected energy cost of uploading S_{up_i}, if uploading of $S_{up_{i-1}}$ finishes at $t_{f_{i-1}}$ and uploading S_{up_i} starts at time slot t_{o_i} and state $X_{t_{o_i}}$, and is given by (17)-(18). Note that we allow the algorithm to decide not to offload or stop offloading if this is to its benefit, by allowing uploading decisions to take the value $t_D + 1$.[2]

For every time slot t_{o_i} and state $X_{t_{o_i}}$, we define the expected cost $V_i(t_{f_{i-1}}, X_{t_{o_i}})$ recursively in (19), for $i = 1, \ldots, K$. $V_i(t_{f_{i-1}}, X_{t_{o_i}})$ can be computed using Dynamic Programming (DP), and it is the minimum between the expected total cost of starting uploading S_{up_i} at time slot t_{o_i} and state $X_{t_{o_i}}$, and the expected cost of postponing that decision to time slot $t_{o_i} + 1$

$$E[V_i(t_{f_{i-1}}, X_{t_{o_i}+1})|X_{t_{o_i}}] = \sum_{X_{t_{o_i}+1} \in \mathcal{T}_i} Pr[X_{t_{o_i}+1}|X_{t_{o_i}}] V_i(t_{f_{i-1}}, X_{t_{o_i}+1}),$$

where $\mathcal{T}_i$ is the set of states reachable after running the channel for $t_{o_i} + 1$ time slots. Note that (19) implies a *policy*, that

[1] We set $t_{f_0} = t_R - 1$.

[2] Equations (9), (10), (12), (13) have been set up to reflect this.

$$\hat{E}_{off_i}(S_{up_i}, T_{up_i}, t_{o_i}) = E_{tr}[\min\{t_D, t_{o_i} + T_{up_i} - 1\} - \min\{t_D, t_{o_i} - 1\}] \quad (9)$$

$$\hat{E}_{off_K}(S_{up_K}, T_{up_K}, t_{o_K}) = E_{tr}[\min\{t_D, t_{o_K} + T_{up_K} - 1\} - \min\{t_D, t_{o_K} - 1\}] + E_{rc}[\min\{t_D, t_{o_K} + T_{rest} - 1\} - \min\{t_D, t_{o_K} + T_{exec} - 1\}] \quad (10)$$

$$E_{off_i}(S_{up_i}, X_{t_{o_i}}) = \sum_{T_{up_i} = \frac{S_{up_i}}{B_{max}}}^{\frac{S_{up_i}}{B_{min}}} P_{t_{o_i}}(S_{up_i}, T_{up_i}, X_{t_{o_i}}) \hat{E}_{off_i}(S_{up_i}, T_{up_i}, t_{o_i}) \quad (11)$$

$$\hat{E}_{L_i}(T_{up_i}, t_{f_{i-1}}, t_{o_i}) = \begin{cases} 0, & t_{o_i} \le t_L - T_{up_i} \text{ or } t_{f_{i-1}} \ge t_D \\ \frac{\min\{t_D, t_{o_i} + T_{up_i} - 1\} - \max\{t_L, t_{f_{i-1}} + 1\} + 1}{T_L} E_L, & \text{otherwise} \end{cases} \quad (12)$$

$$\hat{E}_{L_K}(T_{up_K}, t_{f_{K-1}}, t_{o_K}) = \begin{cases} 0, & t_{o_i} \le t_L - (T_{up_i} + T_{rest}) \text{ or } t_{f_{i-1}} \ge t_D \\ \frac{\min\{t_D, t_{o_K} + T_{up_K} + T_{rest} - 1\} - \max\{t_L, t_{f_{K-1}} + 1\} + 1}{T_L} E_L, & \text{otherwise} \end{cases} \quad (13)$$

$$E_{L_i}(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) = \sum_{T_{up_i} = \frac{S_{up_i}}{B_{max}}}^{\frac{S_{up_i}}{B_{min}}} P_{t_{o_i}}(S_{up_i}, T_{up_i}, X_{t_{o_i}}) \hat{E}_{L_i}(T_{up_i}, t_{f_{i-1}}, t_{o_i}) \quad (14)$$

$$v_i(t_{f_{i-1}}, X_t) = \begin{cases} 0, & t > t_{f_{i-1}} \ge t_D \\ \min\limits_{t \le t_{o_i} \le t_D + 1} \sum\limits_{X_{t_{o_i}} \in \mathcal{S}_i} Pr[X_{t_{o_i}} | X_{t_i}] \Big(E_{off_i}(S_{up_i}, X_{t_{o_i}}) + E_{L_i}(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) \\ + \sum\limits_{X_{t_{f_i}+1} \in \hat{\mathcal{S}}_i} W_i[X_{t_{o_i}}, X_{t_{f_i}+1}] v_{i+1}(t_{f_i}, X_{t_{f_i}+1}) \Big), & t_{f_{i-1}} < t \le t_D \end{cases} \quad (15)$$

$$v_K(t_{f_{K-1}}, X_t) = \begin{cases} 0, & t > t_{f_{K-1}} \ge t_D \\ \min\limits_{t \le t_{o_K} \le t_D + 1} \sum\limits_{X_{t_{o_K}} \in \mathcal{S}_K} Pr[X_{t_{o_K}} | X_t] \Big(E_{off_K}(S_{up_K}, X_{t_{o_K}}) \\ + E_{L_K}(S_{up_K}, t_{f_{K-1}}, X_{t_{o_K}}) \Big), & t_{f_{K-1}} < t \le t_D \end{cases} \quad (16)$$

$$g_i(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) = E_{off_i}(S_{up_i}, X_{t_{o_i}}) + E_{L_i}(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) + \sum_{X_{t_{f_i}+1} \in \mathcal{S}_i} W_i[X_{t_{o_i}}, X_{t_{f_i}+1}] V_{i+1}(t_{f_i}, X_{t_{f_i}+1}), \quad i = 1, \ldots, K \quad (17)$$

$$g_K(S_{up_K}, t_{f_{K-1}}, X_{t_{o_K}}) = E_{off_K}(S_{up_K}, X_{t_{o_K}}) + E_{L_K}(S_{up_K}, t_{f_{K-1}}, X_{t_{o_K}}) \quad (18)$$

$$V_i(t_{f_{i-1}}, X_{t_{o_i}}) = \begin{cases} 0, & t_{o_i} > t_{f_{i-1}} \ge t_D \\ \frac{t_D - \max\{t_{f_{i-1}} + 1, t_L\} + 1}{T_L} E_L, & t_{o_i} \ge t_D > t_{f_{i-1}} \\ \min\{g_i(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}), E[V_i(t_{f_{i-1}}, X_{t_{o_i}+1}) | X_{t_{o_i}}]\}, & t_D > t_{o_i}. \end{cases}, \quad i = 1, \ldots, K \quad (19)$$

dictates whether at any time t_{o_i} and state $X_{t_{o_i}}$ the algorithm should start uploading S_{up_i} (if the min is attained by g_i), or should otherwise wait.

We prove optimality by (reverse) induction. It is well known (e.g., Theorem 1.7 in [30]) that policy V_K is *optimal*, i.e., solves the original problem (16), since

$$v_K(t_{f_{K-1}}, X_{t_K}) = V_K(t_{f_{K-1}}, X_{t_K}), \ \forall t_K > t_{f_{K-1}}, X_{t_K}. \quad (20)$$

Hence the following holds:

Lemma 1. [30] *The optimal time for starting uploading S_{up_K} is*

$$t^*_{o_K} = \underset{t_{f_{K-1}} < t_{o_K} \le t_D}{\arg\min} \{V_K(t_{f_{K-1}}, X_{t_{o_K}}) = g_K(S_{up_K}, t_{f_{K-1}}, X_{t_{o_K}})\} \quad (21)$$

Assuming that decisions $t^*_{o_K}$, $t^*_{o_{K-1}}$, ..., $t^*_{o_{i+1}}$ are optimal, i.e.,

$$v_k(t_{f_{k-1}}, X_{t_k}) = V_k(t_{f_{k-1}}, X_{t_k}),\ \forall t_k > t_{f_{k-1}}, X_{t_k} \quad (22)$$

holds for $k = K, K-1, \ldots, i+1$, we prove that the ith decision $t^*_{o_i}$ of MultiOpt is also optimal. Note that (15) becomes (23). But then, Theorem 1.7 in [30] can be applied again, to show

Lemma 2. [30] *The optimal time for starting uploading S_{up_i} is*

$$t^*_{o_i} = \underset{1 \le t_{o_i} \le t_D}{\arg\min} \{V_i(t_{f_{i-1}}, X_{t_{o_i}}) = g_i(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}})\} \quad (24)$$

Lemmata 1 and 2 imply that the on-line algorithm MultiOpt (Algorithm 1) is optimal for general Markovian channels.

Algorithm 1 MultiOpt (Multi-decision online Optimal)

Input: Local execution starting time t_L, local execution energy E_L, job deadline t_D, and job sizes S_{up_1}, S_{up_2}, ..., S_{up_K}.

```
i = 1
for all t = 1, ..., t_D do
    if job finished uploading then
        Break
    end if
    if currently uploading then
        Continue
    end if
    if min in (19) is g_i then ▷ part i − 1 has been uploaded but i has not started uploading
        start uploading part i
        i = i + 1
    end if
end for
```

6. Simulation Results

In this section, computer simulation is used to study the performance of the proposed K-Part offloading algorithm for $K = 2$, 3 and 4. For comparison, we also plot the *Offline Bound* given in Section 3, and performance of *Local Execution* and three other algorithms, referred to as *OnOpt Algorithm*, *Immediate Offloading*, and *Multi Threshold*. These algorithms all employ concurrent local execution (CLE) to ensure that job execution time constraints are satisfied. The Local Execution algorithm executes the entire job locally without doing any offloading. The OnOpt Algorithm is an online algorithm proposed in [25] that finds the optimum offloading decision to minimize the expected energy consumption of the mobile device. It is the special version of MultiOpt when $K = 1$. For the Immediate Offloading algorithm, a job is offloaded immediately at the release time if $S/B_{max} + T_{rest} \le T_D$; otherwise, the job is executed locally without offloading since offloading cannot be completed before the job deadline even with contiguous best wireless channel states. For the Multi Threshold algorithm, uploading for the first piece starts at the first time slot when the channel condition is above a given threshold, if the remaining time before the job completion deadline is at least $S_{up}/B_{max} + T_{rest}$; otherwise no offloading is performed for the entire job. For the Gilbert-Elliot channel, any threshold between the good and bad states can be used, i.e., uploading starts at the first time slot with the good channel state. After the $(i-1)$th piece is uploaded, uploading for the ith piece starts as soon as the channel state becomes above the threshold, if the remaining time before the job completion deadline is no less than $\sum_{k=i}^{K} S_{up_k}/B_{\max} + T_{rest}$; otherwise, uploading is stopped. In both the Immediate Offloading and the Multi Threshold 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 $t_L - 1$, i.e., they ensure that the job deadline is satisfied.

In this simulation, the job size S_{up}, i.e., total amount of data to be uploaded, is split into K equal parts i.e., $S_{up_1} = S_{up_2} = \cdots = S_{up_K} = S_{up}/K$. The default parameters used in the simulations are given as follows. Each time slot is 1 ms. The transmit and receive power is 1 W and 0.5 W, respectively, which means that the transmission and receive energy during each time slot is $E_{tr} = 1$mJ and $E_{rc} = 0.5$mJ, respectively. In our offloading results we have used the well-known Gilbert-Elliot channel model, that is often used to model random wireless channels [22] [24] [31] [32] [33] [34]. This model is often used to characterize burst noise effects in wireless links, where the channel can abruptly transition between good and bad conditions. This is a good test for computation offloading algorithms with hard execution time constraints, since the channel may be less predictable than those where channel conditions are more correlated and predictable. We assume that $P_{BB} = 1 - P_{GG}$ for the channel state transition probabilities. In this case, $P_{GB} = P_{BB}$, $P_{BG} = P_{GG}$, the equilibrium channel state probabilities are given by $P_g = P_{GG}$ and $P_b = P_{BB}$, and P_{GG} can be used as a measure of the average channel quality. The data transmission rates are $B_b = 1$Mbps and $B_g = 10$Mbps, or $B_b = 1$kb per time slot and $B_g = 10$kb per time slot. Note that in the two-state channel model, $B_{max} = B_g$ and $B_{min} = B_b$. In the results below, each value of average energy consumption is obtained after repeating the simulation for 1500 runs.

6.1. Simulation Set 1

In this section we consider a job with $D = 10$M CPU cycles and $T_D = 60$ time slots. The local execution energy per CPU cycle is $v_l = 2 \times 10^{-6}$mJ and the local computation power is $f_l = 1$M CPU cycles per time slot [35, 36]. 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 of the local processing. The download time T_{down} is assumed to be 1 time slot.

$$v_i(t_{f_{i-1}}, X_t) = \begin{cases} 0, & t > t_{f_{i-1}} \geq t_D \\ \min\limits_{t \leq t_{o_i} \leq t_D+1} \sum\limits_{X_{t_{o_i}} \in \mathcal{S}_i} Pr[X_{t_{o_i}} | X_{t_i}] \Big(E_{off_i}(S_{up_i}, X_{t_{o_i}}) + E_{L_i}(S_{up_i}, t_{f_{i-1}}, X_{t_{o_i}}) \\ + \sum\limits_{X_{t_{f_i}+1} \in \hat{\mathcal{S}}_i} W_i[X_{t_{o_i}}, X_{t_{f_i}+1}] V_{i+1}(t_{f_i}, X_{t_{f_i}+1}) \Big), & t_{f_{i-1}} < t \leq t_D \end{cases} \tag{23}$$

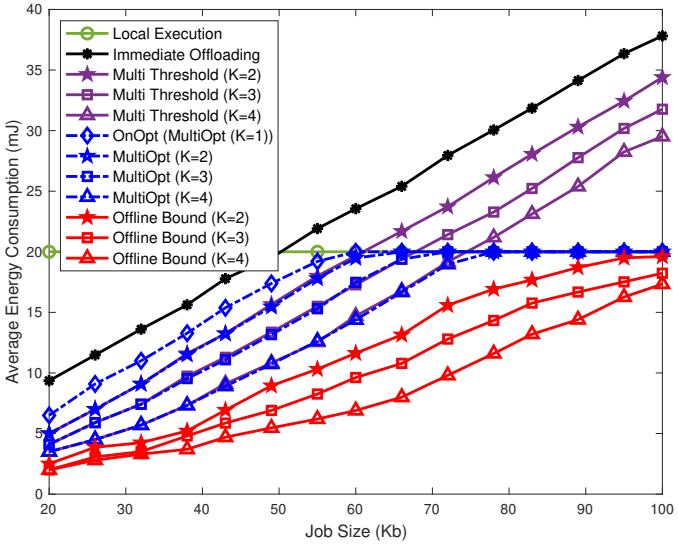

Figure 3: Average energy consumption versus data size S_{up}: $P_{GG} = 0.2$

Figure 3 shows the average energy consumption of the mobile device as the data size S_{up} increases. The energy used by Local Execution is constant for all S_{up}. When S_{up} is smaller, the delay constraint is less stringent, and it is more likely for offloading (without local execution) to meet the delay constraint due to a shorter channel uploading time. In this case, the Multi Threshold and MultiOpt algorithms have approximately the same average energy consumption, since it is more likely for the MultiOpt algorithm to decide starting the uploading as soon as the channel state is good, making it to start the upload at the same time slot with the Multi Threshold algorithm. Compared to OnOpt, the energy consumption of MultiOpt with $K > 1$ is lower, indicating that splitting the job uploading into multiple pieces brings more flexibility that helps the mobile device to avoid uploading during bad channel states; on the other hand, since OnOpt uploads the job continuously, its uploading is more likely to encounter bad channel states, and therefore, takes a longer time and consumes more energy. The Immediate Offloading algorithm consumes higher energy as compared to the other algorithms, because there is a certain probability to encounter bad channel state at the job release time and the following time slots, and the probability becomes higher when P_{GG} is lower. As S_{up} increases, a longer time is needed for wireless transmissions, and the offline bound and the MultiOpt algorithms may decide not to offload, resulting in the same energy consumption as Local Execution, while the Immediate Offloading and Multi Threshold algorithms waste energy consumption by offloading unnecessarily and result in much higher energy consumption.

Comparing the MultiOpt algorithm with $K = 1$ (i.e., the OnOpt algorithm), 2, 3, and 4, we can see that splitting the job into multiple pieces helps reduce the energy consumption of the mobile device, since doing this can avoid uploading during some bad channel states, provided the job completion deadline can be satisfied.

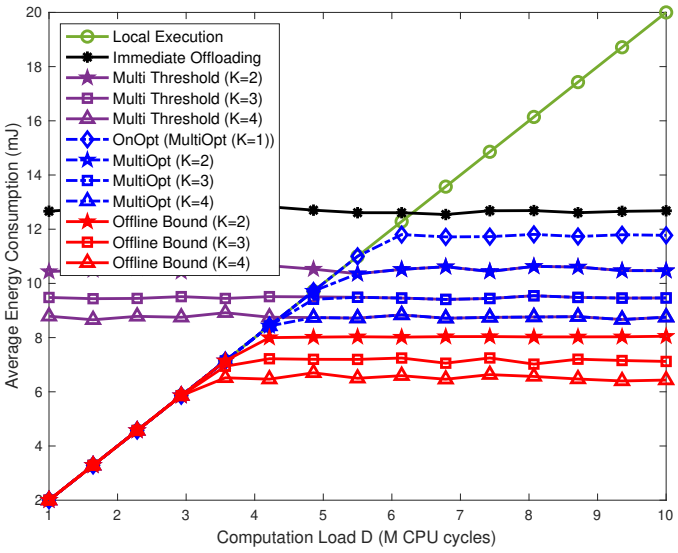

Figure 4: Average energy consumption versus computation load D: $P_{GG} = 0.5$

Figure 4 shows the average energy consumption of the mobile device versus the amount of computation load D. Deadline T_D is set to 40 time slots. When D is small, the MultiOpt algorithm (including OnOpt) does not offload because the local execution energy is low and less than the energy needed to upload the data. As D increases, the energy required for local execution increases, and it becomes more likely that offloading consumes less mobile device energy than local execution. The energy consumption for MultiOpt becomes constant when D is sufficiently large. This is because the delay constraint is relatively loose in the simulated system, which allows offloading to be completed before t_L. Therefore, when D is relatively large, the energy consumption is the same as the energy consumption for wireless transmissions, which does not depend on D. The figure also shows that MultiOpt can save mobile device energy by splitting the job into multiple pieces and uploading separately.

6.2. Simulation Set 2

In this set of results, we use the application parameters for x264 (H.264) encoding from [37], and consider a job with

$S_{up} = 60$Kb, $D = 18$M CPU cycles, and $T_D = 60$ time slots. The local execution energy per CPU cycle is $v_l = 1.5 \times 10^{-6}$mJ and the local computation power is $f_l = 600$ M CPU cycles per second or $f_l = 0.6$ M CPU cycles per time slot. Therefore, the local execution time is $T_L = D/f_l = 30$ time slots, and the local energy consumption $E_L = v_l D = 27$mJ. The remote execution time T_{exec} is 3 time slots, and the download time T_{down} is 1 time slot.

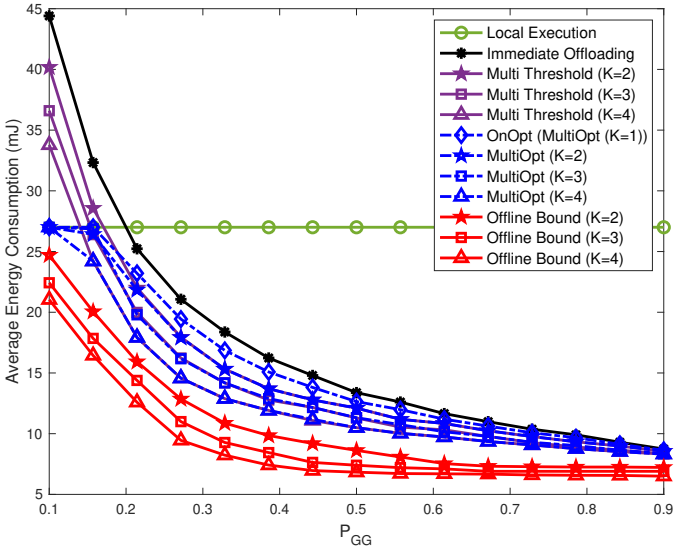

Figure 5: Average energy consumption versus P_{GG}

Figure 5 shows the average energy consumption of different algorithms as P_{GG} varies. The offline bound is close to the energy consumption of Local Execution only when P_{GG} is close to 0, in which case the channel is almost always in the bad state and local execution is almost always the optimum choice. The Immediate Offloading and Multi Threshold algorithms result in much higher energy consumption when P_{GG} is small. Since the channel is in the bad state in most time slots, uploading data requires a long time. Therefore, there is a high probability that offloading cannot meet the delay constraint and/or consumes high energy; furthermore, due to the long uploading time, there may be a long overlap time between offloading and local execution. As a result, energy is unnecessarily wasted in the Immediate Offloading and Multi Threshold algorithms by performing offloading. As P_{GG} increases, the possibility that offloading can meet the deadline increases, so that less local execution is performed, and the total energy consumption decreases for all the offloading algorithms. The Multi Threshold algorithm consumes slightly lower energy than Immediate Offloading. By delaying the uploading (of each piece of the job) until the channel state becomes good, it reduces the transmission time and saves energy consumption. The difference is more obvious when P_{GG} is smaller, since the probability of encountering bad channel states is higher.

When P_{GG} is low, the MultiOpt (including OnOpt) algorithm chooses to not offload, and therefore, results in the same energy consumption as Local Execution; and when P_{GG} is larger, the algorithm more likely chooses to offload, since channel conditions become better and a shorter time and less energy is needed to offload. Figure 5 also shows that, the energy consumption of MultiOpt is lower when K is larger, if the mobile device decides to offload, since splitting the job into more pieces brings more flexibility that helps the mobile device avoid transmissions in bad channel conditions.

By comparing the MultiOpt and Multi Threshold algorithms, we can see that for given K, when P_{GG} is relatively large, the two algorithms consume almost the same energy. This is because the channel condition in general is good, so that the time required for uploading the data is relatively short, and the time required to complete offloading is much less than T_D. For the MultiOpt algorithm, if the decision is to offload the next piece of the job, it is most likely the first time slot with a good channel state, which is the same as the Multi Threshold algorithm. The gap between the MultiOpt algorithm and the offline bound is due to the fact that the MultiOpt algorithm can only use statistical channel information, while the offline bound has knowledge of the channel states of all future time slots.

For all the offloading solutions, the mobile device energy consumption decreases as P_{GG} increases, since the probability of having the good channel state increases, which reduces the time needed to upload the data and makes it more likely for offloading to meet the delay constraint and consume less energy. When P_{GG} is relatively small, the energy decreases fast as P_{GG} increases; and when P_{GG} is sufficiently large, further increasing P_{GG} does not help reduce the energy consumption significantly, since each piece of the job has to be uploaded in consecutive time slots.

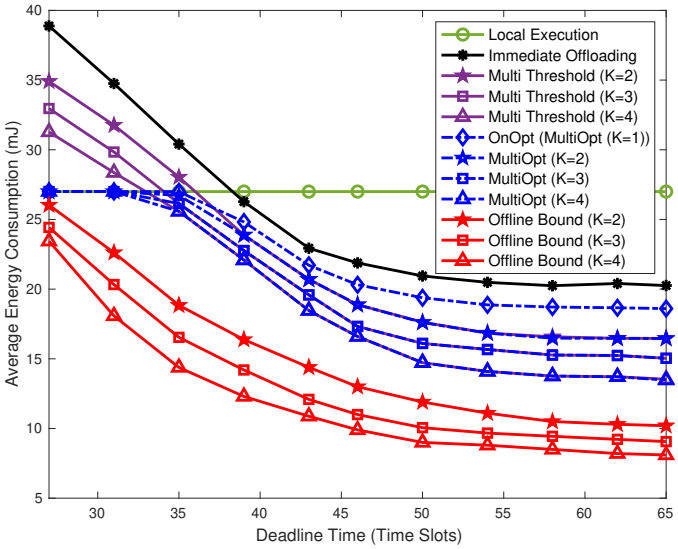

Figure 6: Average energy consumption versus T_D: $P_{GG} = 0.3$

Figure 6 shows the average energy consumption of the algorithms as the job deadline T_D changes. When T_D is small, the MultiOpt (including OnOpt) algorithm decides to not offload most of the time, resulting in the same energy consumption as Local Execution; the offline bound result in lower energy consumption than Local Execution, since it foresees the future channel states and can decide to offload at a future state; while

Immediate Offloading and Multi Threshold most likely result in concurrent offloading and local execution, since offloading cannot meet the delay constraint, and therefore, result in higher energy consumption than Local Execution. The Multi Threshold algorithm achieves some lower energy consumption than the Immediate Offloading algorithm by delaying the offloading until the first time slot with the good channel state.

As T_D increases, more time is available to offload before triggering local execution, resulting in even lower energy consumption for all the offloading algorithms. When T_D is sufficiently large, all the offloading algorithms can achieve lower average energy consumption than using Local Execution. For given K, the MultiOpt and the Multi Threshold algorithms result in the same average energy consumption.

For the offline bound, a loose latency constraint helps it find a better offloading time so that fewer time slots are needed to complete the required transmissions. Ideally, the minimum energy consumption is achieved if there are six consecutive time slots with good channel states before t_L. In general, a larger T_D increases the possibility of having a shorter time interval to complete the uploading, thus reducing the energy consumption. When T_D is small, increasing T_D helps reduce the energy consumption significantly; and when T_D is relatively large, further increasing T_D does not help reduce the energy consumption, since it is almost always possible to complete uploading with six time slots before t_L.

7. Conclusions

This paper has considered mobile computation offloading when concurrent local execution (CLE) is used to enforce task execution time constraints. During task offloading, mobile device energy may sometimes be reduced by segmenting the task upload into multiple parts, rather than doing a conventional contiguous task upload. The advantage of this is that the mobile device can dynamically adapt to changes in channel conditions during the offload. The paper considered the case where the upload is segmented into K parts; in this case, K separate upload initiation decisions are needed. The proposed algorithms do this, while ensuring that hard task deadlines are always satisfied.

The paper analyzed the case of random Markovian wireless channels. An online energy-optimal computation offloading algorithm, MultiOpt, was introduced, and was shown to be optimal in terms of expected energy consumption using Markovian decision process stopping theory. Since the computational complexity of MultiOpt can be significant, simpler and more computationally efficient heuristics, that always satisfy hard task execution deadlines, may be used. The paper introduced two such heuristics, the Immediate Offloading, and Multi Threshold algorithms. The mobile energy use of MultiOpt was compared to these heuristics, as well as to local execution without offloading. Simulation results showed that MultiOpt performs significantly better when compared to the proposed heuristics, as well as when K increases.

References

[1] H. Ba, W. Heinzelman, C.-A. Janssen, J. Shi, Mobile computing - A green computing resource, in: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC), 2013, pp. 4451–4456. doi:10.1109/WCNC.2013.6555295.

[2] G. Huerta-Canepa, 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, 2010, p. 6. doi:10.1145/1810931.1810937.

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

[4] B.-G. Chun, 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, R. Cáceres, The Case for VM-Based Cloudlets in Mobile Computing, IEEE Pervasive Computing 8 (4) (2009) 14–23. doi:10.1109/MPRV.2009.82.

[6] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: making smartphones last longer with code offload, in: Proceedings of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys), 2010, pp. 49–62. doi:10.1145/1814433.1814441.

[7] K. Kumar, Y.-H. Lu, Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?, IEEE Computer 43 (4) (2010) 51–56.

[8] Y. Wen, W. Zhang, 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), 2012, pp. 2716—2720. doi:10.1109/INFCOM.2012.6195685.

[9] O. Muñoz, A. Pascual-Iserte, J. Vidal, Optimization of Radio and Computational Resources for Energy Efficiency in Latency-Constrained Application Offloading, IEEE Transactions on Vehicular Technology 64 (10) (2015) 4738–4755. doi:10.1109/TVT.2014.2372852.

[10] S. Sardellitti, G. Scutari, S. Barbarossa, Joint Optimization of Radio and Computational Resources for Multicell Mobile-Edge Computing, IEEE Transactions on Signal and Information Processing over Networks 1 (2) (2015) 89–103. doi:10.1109/TSIPN.2015.2448520.

[11] X. Chen, L. Jiao, W. Li, X. Fu, Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing, IEEE/ACM Transactions on Networking 24 (5) (2016) 2795–2808. doi:10.1109/TNET.2015.2487344.

[12] R. Kaewpuang, D. Niyato, P. Wang, E. Hossain, A Framework for Cooperative Resource Management in Mobile Cloud Computing, IEEE Journal on Selected Areas in Communications 31 (12) (2013) 2685–2700. doi:10.1109/JSAC.2013.131209.

[13] X. Chen, Decentralized Computation Offloading Game for Mobile Cloud Computing, IEEE Transactions on Parallel and Distributed Systems 26 (4) (2015) 974–983. doi:10.1109/TPDS.2014.2316834.

[14] S. Kosta, A. Andrius, P. Hui, R. Mortier, X. Zhang, ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in: Proceedings of IEEE International Conference on Computer Communications (INFOCOM), 2012, pp. 945–953. doi:10.1109/INFCOM.2012.6195845.

[15] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, F. Bai, Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing, in: Proceedings of IEEE International Conference on Computer Communications (INFOCOM), 2015, pp. 1894–1902. doi:10.1109/TMC.2017.2679712.

[16] M.-H. Chen, B. Liang, M. Dong, Joint offloading decision and resource allocation for multi-user multi-task mobile cloud, in: Proceedings of IEEE International Conference on Communications (ICC), 2016, pp. 1–6. doi:10.1109/ICC.2016.7510999.

[17] E. Meskar, T. D. Todd, D. Zhao, G. Karakostas, Energy Aware Offloading for Competing Users on a Shared Communication Channel, IEEE Transactions on Mobile Computing 16 (1) (2017) 87–96. doi:10.1109/TMC.2016.2538227.

[18] E. Meskar, T. D. Todd, D. Zhao, G. Karakostas, Energy efficient offloading for competing users on a shared communication channel, in: 2015 IEEE International Conference on Communications, ICC 2015,

London, United Kingdom, June 8-12, 2015, 2015, pp. 3192–3197. doi:10.1109/ICC.2015.7248815.

[19] S. Jošilo, G. Dán, A game theoretic analysis of selfish mobile computation offloading, in: Proceedings of IEEE International Conference on Computer Communications (INFOCOM), 2017, pp. 1–9. doi:10.1109/INFOCOM.2017.8057148.

[20] H. Cao, J. Cai, Distributed Multiuser Computation Offloading for Cloudlet-Based Mobile Cloud Computing: A Game-Theoretic Machine Learning Approach, IEEE Transactions on Vehicular Technology 68 (1) (2018) 752–764. doi:10.1109/TVT.2017.2740724.

[21] H. Lagar-Cavilla, N. Tolia, E. D. Lara, M. Satyanarayanan, D. O'Hallaron, Interactive resource-intensive applications made easy, in: ACM/IFIP/USENIX International Conf. Middleware, 2007, pp. 143–163.

[22] W. Zhang, Y. Wen, D. O. Wu, Collaborative Task Execution in Mobile Cloud Computing Under a Stochastic Wireless Channel, IEEE Transactions on Wireless Communications 14 (1) (2015) 81–93.

[23] Y. Liu, 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), 2014, pp. 1868–1873. doi:10.1109/WCNC.2014.6952554.

[24] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, D. O. Wu, Energy-Optimal Mobile Cloud Computing under Stochastic Wireless Channel, IEEE Transactions on Wireless Communications 12 (9) (2013) 4569–4581. doi:10.1109/TWC.2013.072513.121842.

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

[26] A. Hekmati, P. Teymoori, T. D. Todd, D. Zhao, G. Karakostas, Optimal multi-decision mobile computation offloading with hard task deadlines, in: IEEE Symposium on Computers and Communications (ISCC), 2019.

[27] C. You, K. Huang, H. Chae, Energy Efficient Mobile Cloud Computing Powered by Wireless Energy Transfer, IEEE Journal on Selected Areas in Communications 34 (5) (2016) 1757–1771. doi:10.1109/JSAC.2016.2545382.

[28] M. Chiang, T. Zhang, Fog and IoT: An Overview of Research Opportunities, IEEE Internet of Things Journal 3 (6) (2016) 854–864. doi:10.1109/JIOT.2016.2584538.

[29] C. M. Grinstead, J. L. Snell, Chapter 11 - markov chains, in: Grinstead and Snell's Introduction to Probability, 2006, pp. 405–470.

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

[31] E. N. Gillbert, Capacity of a burst noise channel, Bell Syst. Tech. J. 39 (1960) 1253—1266.

[32] E. O. Elliott, Estimates of error rates for codes on burst-noise channels, Bell Syst. Tech. J. 42 (1963) 1977—1997.

[33] L. A. Johnston, V. Krishnamurthy, Opportunistic file transfer over a fading channel: A POMDP search theory formulation with optimal threshold policies, IEEE Transactions on Wireless Communications 5 (2) (2006) 394–405. doi:10.1109/TWC.2006.1611063.

[34] M. Zafer, E. Modiano, Minimum energy transmission over a wireless fading channel with packet deadlines, in: Proceedings of IEEE International Conference on Decision and Control, 2007, pp. 1148–1155. doi:10.1109/CDC.2007.4434683.

[35] M. Nir, A. Matrawy, 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.

[36] D. Huang, P. Wang, D. Niyato, A dynamic offloading algorithm for mobile computing, Wireless Communications, IEEE Transactions on 11 (6) (2012) 1991–1995.

[37] A. P. Miettinen, J. K. Nurminen, Energy efficiency of mobile clients in cloud computing, in: Proceedings of the 2nd USENIX conference on hot topics in cloud computing, Berkeley, CA, USA, 2010.