Optimal Multi-Decision Mobile Computation Offloading With Hard Task Deadlines

Arvin Hekmati*, Peyvand Teymoori*, Terence D. Todd*, Dongmei Zhao* and George Karakostas†
*Department of Electrical and Computer Engineering
†Department of Computing & Software
McMaster University
Hamilton, Ontario, CANADA
Email: {hekmatia,teymoorp,todd,dzhao,karakos}@mcmaster.ca

***Abstract*—Multi-decision mobile computation offloading occurs when a task to be remotely executed is uploaded in separate parts. Since the upload is partitioned, separate decisions are needed to determine the best time to initiate each upload. The multi-decision problem is considered for the case where execution completion times are subject to hard deadline constraints and where task offloads occur over a Markovian wireless channel. An online energy-optimal computation offloading algorithm, MultiOpt (Multi-decision online Optimum), is introduced, whose optimality is proven using Markovian stopping theory. The paper presents results using the Gilbert-Elliott channel model, where task completion time probabilities can be efficiently computed using Dynamic Programming. Although the proposed algorithm is proven to be energy optimal, its performance is also compared to four others, namely, Immediate Offloading, Channel Threshold, Local Execution, as well as optimal single-part 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.**

***Index Terms*—Green wireless communications, cloud computing, mobile computation offloading, energy efficiency, hard task deadline constraints.**

I. Introduction

Mobile computation offloading can be used to reduce mobile device energy by offloading task execution to remote cloud servers [1]. A lot of recent work has considered mobile computation offloading when the mobile device must interact with the cloud over stochastic transmission channels and/or network conditions. In [2], for example, an energy model considers both mobile computation and communication energy components using statistical inputs, assuming a static wireless channel. In this work, prediction was used to address random network conditions. Reference [3] considered optimal energy mobile cloud computing assuming random wireless channels but without hard execution deadlines. Reference [4] flagged execution time constraints as an important issue for many interactive applications and the problem of achieving this under stochastic channel conditions was highlighted. In [5], CPU frequency scheduling and transmit power control was used to ensure that task deadlines are met. A parameter was defined, however, that permits deadlines to be violated. As in our case, this reference presented results using the well-known Gilbert-Elliot channel model. Reference [6] proposed an on-line mobile computation offloading algorithm, OnOPT, that was shown to be energy optimal and satisfies hard deadline constraints. This algorithm is used for performance comparisons in our paper since it represents the best energy performance that can be obtained with single-part offloading.

Research supported by the last three authors' NSERC Discovery Grants

Our paper considers multi-decision mobile computation offloading. This occurs when a task to be remotely executed is uploaded in separate parts rather than as a single contiguous multi-packet upload. Multi-decision offloading can be used to reduce mobile energy use when wireless channel conditions change during the computation offload. Since the upload is partitioned into parts, separate decisions are needed to determine the best time to initiate each upload. The multi-decision problem is considered when task execution completion times are subject to hard deadline constraints, i.e., the upload decisions must ensure that task completion deadlines are always satisfied. Hard deadlines are guaranteed by permitting *simultaneous* remote and local task execution, as needed [6]. The paper considers the case for Markovian wireless channel models. An online energy-optimal computation offloading algorithm, MultiOpt (Multi-decision online Optimum), is introduced, whose optimality is proven using optimal Markovian stopping theory. The exposition in the paper considers the two part offloading case, but the methodology is easily extended to an arbitrary number of offloading parts. To assess the performance of the algorithm under harsh burst noise conditions, results are presented for the Gilbert-Elliott channel model, where task completion time probabilities can be efficiently computed using Dynamic Programming. The proposed algorithm performance is compared to four others, namely, Immediate Offloading, Channel Threshold, Local Execution, as well as the optimal single-part offloading (i.e., OnOpt). Results show that the proposed algorithm can significantly improve mobile device energy consumption compared to the other approaches, while guaranteeing hard task execution deadlines.

II. 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. Our discussion will focus on single job offloads, but each job could be a sub-task associated with multiple local/remote job execution components [7] [8]. We assume that each job can be split into multiple 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. To simplify the presentation, we will describe the two part case, but the procedure is easily extended to multiple parts. Accordingly, the number of bits to be uploaded are given by S_{up_1}, S_{up_2} for each part, respectively, for a total of $S_{up} = S_{up_1} + S_{up_2}$ bits. S_{down} bits are transmitted through the downlink channel when downloading job results from the cloud. Figure 1 defines the timing parameters used in this work; time is discretized, i.e., quantized into equal length *time slots*. Note that the time slot duration is defined to accommodate the channel propagation model discussed in Section III, and may contain multiple packet transmission times on the channel. Each job is released at time slot t_r; for convenience, we assume that $t_r = 1$. The job execution must have been completed by a *hard deadline* t_D, i.e., the job execution results *must* be available at the mobile device by time slot $t_D + 1$.

A. Local Execution

Using the model described in [2], 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 *local execution time* 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 time is given by $t_L = t_D - T_L + 1$.

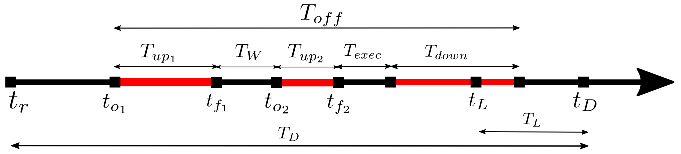

Fig. 1. Job offloading timing parameters

B. Remote Execution

In the case of offloading a job, the uploading data are split into two parts which are transmitted to the server sequentially. Uploading the first and second part takes T_{up_1} and T_{up_2} time slots respectively; T_W is the elapsed time between the two uploads. Upon its release, the job is assigned a server execution time T_{exec} and a resulting downloading time T_{down} (both deterministic) by the cloud server; both are communicated to the mobile device (or are prescribed by, say, the contractual agreement between the user of the device and the cloud server operator), and their total time is

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

Hence, the total offloading time is $T_{off} = T_{up_1} + T_W + T_{up_2} + T_{exec} + T_{down}$, as shown in Figure 1 (note that t_{o_1}, t_{f_1} and t_{o_2}, t_{f_2} are the starting and finishing times of the uploading of the two job parts, respectively). It is assumed that the mobile device transmits a fixed power and uses bit rate adaptation to accommodate random variations in the uplink channel conditions. As a result, T_{up_1} and T_{up_2} are random variables, 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. As in most of previous work, 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 base station at the start of the time slot.

III. Markovian Channel 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 $\mathbb{P} = [P_{i,j}]$, 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. This new Markov chain is referred to as a *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 $TDAMC_1$ and $TDAMC_2$ below), endowed with energy costs and absorbing states.

The part of the TDAMC which 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 Gilber-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 $P_{GG}, P_{GB}, P_{BG}, P_{BB}$. In each time slot, $TDAMC_1$ transitions to a new state in accordance with these transition proba-

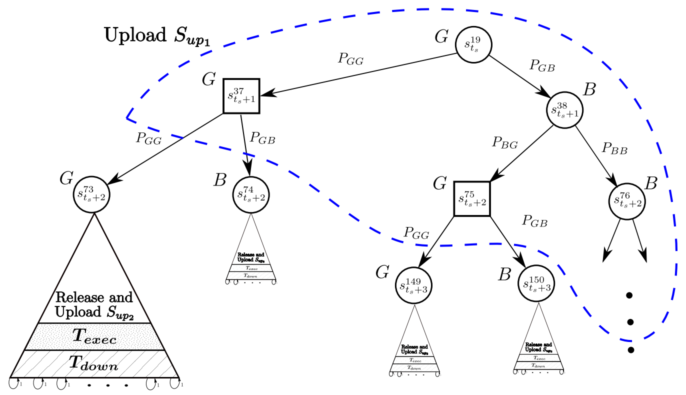

Fig. 2. $TDAMC_1$ when offloading S_{up_1} starts at time t_s.

bilities. For clarity, each state s^a_t 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 $s^{19}_{t_s}$. 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 $s^{37}_{t_s+1}, s^{75}_{t_s+2}$, 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 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 is depicted in Figure 2 as subtrees hanging from states $s^{73}_{t_s+2}, s^{74}_{t_s+2}, s^{149}_{t_s+3}, s^{150}_{t_s+3}$). 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}, is solved in Section IV. Then the energy cost of each subtree is the optimal expected (over all paths) cost of completing offloading, when uploading S_{up1} 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 treats states $X_{t_{f_1}+1}$ as absorbing states, each with cost equal to the energy cost of its subtree.

Similarly to [6], the probability of uploading S_{up_1} in T_{up_1} time slots, starting at time slot t_{o_1}, and a state $X_{t_{o_1}}$, can be calculated by building $TDAMC_1$, with a set of absorbing states $\mathcal{A}$, and a set of transient states $\mathcal{T}$. Then, the transition matrix can be written [9] as

$$\mathbb{P} = \begin{bmatrix} Q & R \\ \mathbf{0} & I_{\mathcal{A}} \end{bmatrix}, \tag{1}$$

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

The theory of absorbing Markov chains implies that various statistics can be computed by forming the fundamental matrix $N = (I - Q)^{-1}$, where $N[i,j]$ gives the expected number of times that $TDAMC_1$ is in transient state j if the system is started in transient state i. Given the structure of $TDAMC_1$, N can be easily decomposed and calculated as in [6], since the particular structure of matrices Q, N, N^{-1} is the same simple one as in [6]. The absorption probabilities matrix W_1 for all absorbing states is given by

$$W_1 = NR, \tag{2}$$

where W_1 is a $|\mathcal{T}| \times |\mathcal{A}|$ matrix, and $W_1[i,j]$ gives the probability that absorbing state j will be reached when starting in transient state i. Therefore, the probability of uploading the first part with size S_{up_1} in T_{up_1} time slots, starting at time t_{o_1} and state $X_{t_{o_1}}$, is

$$P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) = \textstyle\sum_{j \in \mathcal{S}_1} W_1[X_{t_{o_1}}, j], \tag{3}$$

where $\mathcal{S}_1$ is the set of absorbing states in $TDAMC_1$ reached by a path of length $T_{up_1}+1$ from the root $X_{t_{o_1}}$.

Similarly to $TDAMC_1$, and in order to calculate the expected cost once the uploading of S_{up_2} commences at time slot t_{o_2}, we construct $TDAMC_2$, which tracks the offloading process from t_{o_2} and state $X_{t_{o_2}}$ until offloading is completed. Just like above, the probability of uploading S_{up_2} in T_{up_2} time slots, starting at time t_{o_2} and state $X_{t_{o_2}}$, is

$$P_{t_{o_2}}(S_{up_2}, T_{up_2}, X_{t_{o_2}}) = \textstyle\sum_{j \in \mathcal{S}_2} W_2[X_{t_{o_2}}, j] \tag{4}$$

where $\mathcal{S}_2$ is the set of absorbing states in $TDAMC_2$ reached by a path of length $T_{up_2}+1$ from the root $X_{t_{o_2}}$.

If the uploading of S_{up_1} starts at time slot t_{o_1}, and after noting that $P_{t_{o_1}}(S_{up_1}, T_{up_1}, x) = 0$ when $T_{up_1} < \frac{S_{up_1}}{B_{max}}$ or $T_{up_1} > \frac{S_{up_1}}{B_{min}}$, the expected offloading energy cost when offloading starts at time slot t_{o_1} in state $X_{t_{o_1}}$, is given by equation (5), and the expected local execution cost is given by (6), where E_{tr} is the transmission energy of the mobile device during one time slot. (Equations (5) to (8) appear at the top of the next page.)

The expected energy cost of uploading S_{up_2} in exactly T_{up_2} time slots, and downloading the results in exactly T_{down} time slots, starting at time t_{o_2} with the channel TDAMC in state $X_{t_{o_2}}$, is given by equation (7), 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

$$E_{off_1}(S_{up_1}, X_{t_{o_1}}) = \begin{cases} E_{tr} \sum_{T_{up_1}=\frac{S_{up_1}}{B_{max}}}^{\frac{S_{up_1}}{B_{min}}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) T_{up_1}, & 1 \le t_{o_1} < t_D - \frac{S_{up_1}}{B_{min}} + 1 \\ E_{tr} \left(\sum_{T_{up_1}=\frac{S_{up_1}}{B_{max}}}^{t_D - t_{o_1}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) T_{up_1} + \right. & \\ \left. \sum_{T_{up_1}=t_D - t_{o_1}+1}^{\frac{S_{up_1}}{B_{min}}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}})(t_D - t_{o_1} + 1) \right), & t_D - \frac{S_{up_1}}{B_{min}} + 1 \le t_{o_1} \le t_D \end{cases} \quad (5)$$

$$E_{L_1}(S_{up_1}, X_{t_{o_1}}) = \begin{cases} \sum_{T_{up_1}=t_L - t_{o_1}+1}^{\frac{S_{up_1}}{B_{min}}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) \left(\frac{\min\{t_D+1, t_{o_1}+T_{up_1}\} - t_L}{T_L} E_L \right), & 1 \le t_{o_1} < t_L \\ \sum_{T_{up_1}=\frac{S_{up_1}}{B_{max}}}^{\frac{S_{up_1}}{B_{min}}} P_{t_{o_1}}(S_{up_1}, T_{up_1}, X_{t_{o_1}}) \left(\frac{\min\{t_D+1, t_{o_1}+T_{up_1}\} - t_L}{T_L} E_L \right), & t_L \le t_{o_1} \le t_D \end{cases} \quad (6)$$

$$\hat{E}_{off_2}(S_{up_2}, T_{up_2}, t_{o_2}) = \begin{cases} E_{tr} T_{up_2} + E_{rc} T_{down}, & t_{f_1} < t_{o_2} \le t_D - T_{up_2} - T_{rest} \\ E_{tr} T_{up_2} + E_{rc}\{t_D - (t_{o_2} + T_{up_2} + T_{exec}) + 1\}, & t_D - T_{rest} < t_{o_2} + T_{up_2} \le t_D - T_{exec} \\ E_{tr} T_{up_2}, & t_D - T_{exec} < t_{o_2} + T_{up_2} \le t_D \\ E_{tr}(t_D - t_{o_2} + 1), & t_D < t_{o_2} + T_{up_2} \le t_D + T_{up_2} \end{cases} \quad (7)$$

$$\hat{E}_{L_2}(T_{up_2}, t_{f_1}, t_{o_2}) = \begin{cases} 0, & t_{f_1} < t_{o_2} < t_L - T_{up_2} - T_{rest} \\ \frac{t_{o_2} + T_{up_2} + T_{rest} - t_L}{T_L} E_L, & t_{f_1} < t_L \wedge t_L - T_{up_2} - T_{rest} \le t_{o_2} \le t_D - T_{up_2} - T_{rest} \\ E_L, & t_{f_1} < t_L \wedge t_D - T_{up_2} - T_{rest} < t_{o_2} \le t_D \\ \frac{t_{o_2} + T_{up_2} + T_{rest} - t_{f_1}}{T_L} E_L, & t_{f_1} \ge t_L \wedge t_{f_1} < t_{o_2} \le t_D - T_{up_2} - T_{rest} \\ \frac{t_D - t_{f_1}}{T_L} E_L, & t_{f_1} \ge t_L \wedge t_D - T_{up_2} - T_{rest} < t_{o_2} \le t_D \end{cases} \quad (8)$$

cost is

$$E_{off_2}(S_{up_2}, X_{t_{o_2}}) = \sum_{T_{up_2}=\frac{S_{up_2}}{B_{max}}}^{\frac{S_{up_2}}{B_{min}}} P_{t_{o_2}}(S_{up_2}, T_{up_2}, X_{t_{o_2}}) \hat{E}_{off_2}(S_{up_2}, T_{up_2}, t_{o_2}). \quad (9)$$

Given the finishing time t_{f_1} of uploading S_{up_1}, the local execution energy cost corresponding to the offloading portion, starting with the uploading of S_{up_2} at time t_{o_2} and state $X_{t_{o_2}}$, taking exactly T_{up_2} time slots, and finishing with the downloading of the results, is given in (8). Then the expected local execution energy cost is

$$E_{L_2}(S_{up_2}, t_{f_1}, X_{t_{o_2}}) = \sum_{T_{up_2}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_{t_{o_2}}(S_{up_2}, T_{up_2}, X_{t_{o_2}}) \hat{E}_{L_2}(T_{up_2}, t_{f_1}, t_{o_2}) \quad (10)$$

Note that $E_{off_1} = 0, E_{L_1} = E_L$ for $t_{o_1} \ge t_D + 1$, and $E_{off_2} = 0, E_{L_2} = E_L$ for $t_{o_2} \ge t_D + 1$, i.e., when the first or second part isn't uploaded, respectively.

IV. Optimal Stopping and the MultiOpt (Multi-Decision online Optimal) Algorithm

In this section we use the TDAMC construction of Section III and the theory of optimal stopping for Markov decision processes [10] to define the MultiOpt algorithm, and show that it achieves the optimal expected energy for the mobile device. A high-level description of the algorithm 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 to time slot $t + 1$. Once the uploading of S_{up_1} finishes, the algorithm repeats the same decision process at every time slot (using $TDAMC_2$ to compute expected costs), to determine the time $t^*_{o_2}$ of starting uploading S_{up_2}.

MultiOpt will be optimal only if its first decision $t^*_{o_1} \ge t$, i.e., its starting time of uploading S_{up_1}, coincides with the solution of the following minimization problem (where the choice $t_{o_1} = t_D + 1$ corresponds to no uploading):

$$v_1(X_t) = \min_{t \le t_{o_1} \le t_D + 1} \left\{ \sum_{X_{t_{o_1}} \in \mathcal{S}_1} Pr[X_{t_{o_1}} | X_t] \right.$$

$$\Big(E_{off_1}(S_{up_1}, X_{t_{o_1}}) + E_{L_1}(S_{up_1}, X_{t_{o_1}}) + \sum_{X_{t_{f_1}+1} \in \mathcal{S}_2} W_1[X_{t_{o_1}}, X_{t_{f_1}+1}] v_2(t_{f_1}, X_{t_{f_1}+1})\Big)\Bigg\} \quad (11)$$

where $\mathcal{S}_1$ is the set of states reachable after running the channel for t_{o_1} time slots, $\mathcal{S}_2$ is the set of absorbing states of $TDAMC_1$ rooted at $X_{t_{o_1}}$, and $v_2(t_{f_1}, X_{t_{f_1}+1})$ is the optimal expected energy cost for the rest of the offloading, when S_{up_1} finished uploading at time t_{f_1}, i.e., the cost of the absorbing state $X_{t_{f_1}+1}$ of $TDAMC_1$ (or, equivalently, the corresponding subtree of Figure 2). This optimal cost is the solution of the following optimization problem for $t > t_{f_1}$, when we set $X_t := X_{f_1+1}$:

$$\begin{aligned} v_2(t_{f_1}, X_t) &= \min_{t \le t_{o_2} \le t_D+1} E[g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}})|X_t] \\ &= \min_{t \le t_{o_2} \le t_D+1} \sum_{X_{t_{o_2}} \in \mathcal{T}_1} Pr[X_{t_{o_2}}|X_t] g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}}), \end{aligned} \quad (12)$$

where $\mathcal{T}_1$ is the set of states reachable after running the channel for t_{o_1} time slots, and $g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}})$ is the expected energy cost of uploading S_{up_2} and downloading the results, if uploading of S_{up_1} finishes at t_{f_1} and uploading S_{up_2} starts at time slot t_{o_2} and state $X_{t_{o_2}}$, i.e.,

$$\begin{aligned} g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}}) &= \\ E_{off_2}(S_{up_2}, X_{t_{o_2}}) &+ E_{L_2}(S_{up_2}, t_{f_1}, X_{t_{o_2}}). \end{aligned} \quad (13)$$

For $t > t_D$, $v_2(t_{f_1}, X_t) = 0$ (no uploading of the second part).

Given the first decision $t^*_{o_1}$ of MultiOpt, we show that its second decision $t^*_{o_2}$ solves the optimization problem (12). For every time slot $t_{o_2} > t_{f_1}$ and state $X_{t_{o_2}}$, we define the expected cost $V_2(t_{f_1}, X_{t_{o_2}})$ recursively as follows:

$$V_2(t_{f_1}, X_{t_{o_2}}) = \begin{cases} 0, \text{ if } t_{o_2} > t_{f_1} \ge t_D \\ E_L - \frac{\max\{t_{f_1}, t_L\} - t_L}{T_L} E_L, \text{ if } t_{o_2} \ge t_D > t_{f_1} \\ \min\{g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}}), E[V_2(t_{f_1}, X_{t_{o_2}+1})|X_{t_{o_2}}]\}, \\ \quad \text{if } t_D > t_{o_2}. \end{cases} \quad (14)$$

$V_2(t_{f_1}, X_{t_{o_2}})$ can be computed using Dynamic Programming (DP), and it is the minimum between the expected total cost of starting uploading S_{up_2} at time slot t_{o_2} and state $X_{t_{o_2}}$, and the expected cost of postponing that decision to time slot $t_{o_2} + 1$

$$E[V_2(t_{f_1}, X_{t_{o_2}+1})|X_{t_{o_2}}] = \sum_{X_{t_{o_2}+1} \in \mathcal{T}_2} Pr[X_{t_{o_2}+1}|X_{t_{o_2}}] V_2(t_{f_1}, X_{t_{o_2}+1}),$$

where $\mathcal{T}_2$ is the set of states reachable after running the channel for $t_{o_2} + 1$ time slots. Note that (14) implies a *policy*, that dictates whether at any time t_{o_2} and state $X_{t_{0_2}}$ the algorithm should start uploading (if the $\min$ is attained by g_2), or should otherwise wait. It is well known (e.g., Theorem 1.7 in [10]) that policy V_2 is *optimal*, i.e., solves the original problem (12), since

$$v_2(t_{f_1}, X_t) = V_2(t_{f_1}, X_t), \ \forall t > t_{f_1}, X_t. \quad (15)$$

Hence the following holds:

Lemma 1. **[10]** *The optimal time for starting uploading S_{up_2} is $t^*_{o_2} = \arg\min_{t_{f_1} < t_{o_2} \le t_D}\{V_2(t_{f_1}, X_{t_{o_2}}) = g_2(S_{up_2}, t_{f_1}, X_{t_{o_2}})\}$.*

It remains to prove that the first decision $t^*_{o_1}$ of MultiOpt is also optimal. For any possible choice t_{o_1} for the first decision of MultiOpt, (15) can be applied, and the optimization problem (11) becomes

$$v_1(X_t) = \min_{t \le t_{o_1} \le t_D+1} \Bigg\{ \sum_{X_{t_{o_1}} \in \mathcal{S}_1} Pr[X_{t_{o_1}}|X_t] \Big(E_{off_1}(S_{up_1}, X_{t_{o_1}}) + E_{L_1}(S_{up_1}, X_{t_{o_1}}) + \sum_{X_{t_{f_1}+1} \in \mathcal{S}_1} W_1[X_{t_{o_1}}, X_{t_{f_1}+1}] V_2(t_{f_1}, X_{t_{f_1}+1})\Big)\Bigg\} \quad (16)$$

The expected energy cost of offloading when starting uploading S_{up_1} at time t_{o_1} and state $X_{t_{o_1}}$ is

$$\begin{aligned} g_1(S_{up_1}, X_{t_{o_1}}) = E_{off_1}(S_{up_1}, X_{t_{o_1}}) + E_{L_1}(S_{up_1}, X_{t_{o_1}}) + \\ \sum_{X_{t_{f_1}+1} \in \mathcal{S}_1} W_1[X_{t_{o_1}}, X_{t_{f_1}+1}] V_2(t_{f_1}, X_{t_{f_1}+1}). \end{aligned} \quad (17)$$

For every time slot t_{o_1} and state $X_{t_{o_1}}$, we define the expected cost $V_1(X_{t_{o_1}})$ recursively as follows:

$$V_1(X_{t_{o_1}}) = \begin{cases} E_L, & t_{o_1} \ge t_D \\ \min\Big\{g_1(S_{up_1}, X_{t_{o_1}}), \\ E[V_1(X_{t_{o_1}+1})|X_{t_{o_1}}]\Big\}, & t_{o_1} = 1, \ldots, t_D - 1 \end{cases} \quad (18)$$

$V_1(X_{t_{o_1}})$ can be computed using Dynamic Programming (DP), and it is the minimum between the expected total cost of starting uploading S_{up_1} at time slot t_{o_1} and state $X_{t_{o_1}}$, and the expected cost of postponing that decision to time slot $t_{o_1} + 1$

$$E[V_1(X_{t_{o_1}+1})|X_{t_{o_1}}] = \sum_{X_{t_{o_1}+1} \in \mathcal{S}_3} Pr[X_{t_{o_1}+1}|X_{t_{o_1}}] V_1(X_{t_{o_1}+1}),$$

where $\mathcal{S}_3$ is the set of states reachable after running the channel for $t_{o_1} + 1$ time slots.

In exactly the same way as Lemma 1, one can show that policy V_1 is also *optimal*, i.e., solves the original problem (11), since $v_1(X_t) = V_1(X_t), \ \forall t, X_t$. Hence the following holds:

Lemma 2. **[10]** *The optimal time for starting uploading S_{up_1} is $t^*_{o_1} = \arg\min_{1 \le t_{o_1} \le t_D}\{V_1(X_{t_{o_1}}) = g_1(S_{up_1}, X_{t_{o_1}})\}$.*

Lemmata 1 and 2 imply that the on-line algorithm MultiOpt, given in Algorithm 1, is optimal. Note that this result is true for any Markovian channel.

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} .

```
1: for all $t = 1, \dots, t_D$ do
2:     Case 1: If uploading part 2 is finished then Break
3:     Case 2: If still uploading at $t$ then Continue
4:     Case 3: If uploading part 1 not started then perform check (18); If min is $g_1$ then start uploading part 1.
5:     Case 4: If part 1 has been uploaded but part 2 has not started uploading then perform check (14); If min is $g_2$ then start uploading part 2.
6: end for
```

V. Simulation Results

In this section, computer simulation is used to study the performance of the proposed MultiOpt Algorithm. The Gilbert-Elliot Markovian channel model is used for our examples. This model is commonly used to model harsh channel conditions where burst noise causes the channel to abruptly transition between good and poor channel conditions. This type of channel posses severe problems for computational offloading due to the random fluctuations between these two extremes during the channel offload. The Gilber-Elliot model has two states, which we refer to as Good (G) and Bad (B), with bit rates B_g and B_b, respectively. We set $P_{BB} := 1 - P_{GG}$, in order to use P_{GG} as a measure of the average channel quality, i.e., larger P_{GG} indicates better channel conditions on average.

We compare the energy consumption of MultiOpt to an *offline bound*, the *Local Execution* of the job, and three other algorithms, namely *OnOpt*, *Immediate Offloading*, and *Channel Threshold*. The offline bound finds the optimal uploading times by assuming complete knowledge of all future channel states. The Local Execution of the entire job is done locally at the mobile device, without doing any offloading. The OnOpt Algorithm, proposed in [6], is an online algorithm that finds the optimum offloading start time to minimize the expected energy consumption when the job is uploaded in one part without interruption. The Immediate Offloading algorithm offloads the job immediately at its release time, unless $S_{up}/B_g + T_{rest} > T_D$, i.e., unless offloading cannot be completed before the job deadline even under the best channel conditions, in which case the job is executed locally without offloading. The Channel Threshold algorithm starts the uploading of the first part at the first time slot when the channel condition becomes Good, unless the remaining time before t_D is less than $S_{up}/B_g + T_{rest}$; when uploading the first part is completed (if the decision is to offload), uploading the second part starts as soon as the channel state becomes Good, unless the remaining time before t_D is less than $S_{up_2}/B_g + T_{rest}$. For both the Immediate Offloading and the Channel Threshold algorithms, local execution starts at time slot t_L if offloading is not completed at time slot $t_L - 1$.

We will assume that the total amount of data to be offloaded is split into two equal parts, i.e., $S_{up_1} = S_{up_2} = S_{up}/2$.

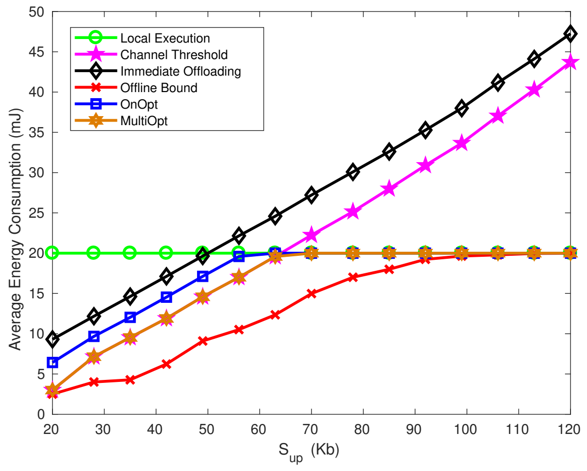

Fig. 3. Average energy consumption versus S_{up}: $P_{GG} = 0.2$

The parameter settings used in the simulations are as follows: Each time slot lasts for 1 ms. 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. The transmission and reception power of the mobile device is 1 W and 0.5 W, respectively, which means that the transmission and reception energy per time slot is $E_{tr} = 1$mJ and $E_{rc} = 0.5$mJ, respectively. The download time T_{down} is 1 time slot. In the results below, the average energy consumption is obtained after repeating the simulation for 10,000 runs.

We first consider a job that needs $D = 10$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 [11], [12]. 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. The remote execution time is $T_{exec} = 1$ time slot. 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, it is more likely for offloading to meet the delay constraint due to shorter channel uploading time. Therefore, the energy consumption of all offloading algorithms is smaller than that of Local Execution. As S_{up} increases, the average energy consumption of the Immediate Offloading and Channel Threshold algorithms keeps increasing, and can be much larger than that of Local Execution, while the average energy consumption of the offline bound, MultiOpt, and OnOpt algorithms increases first and then keeps the same as that of Local Execution as S_{up} becomes large. The Immediate Offloading algorithm has the highest energy consumption among all the algorithms because it always offloads. By delaying the offloading until the first Good channel state, the Channel Threshold algorithm consumes slightly lower average energy than Immediate Offloading, but its average energy consumption still keeps increasing with S_{up}. This is due to the fact that the offloading decision of the Channel Threshold algorithm is most beneficial in case of

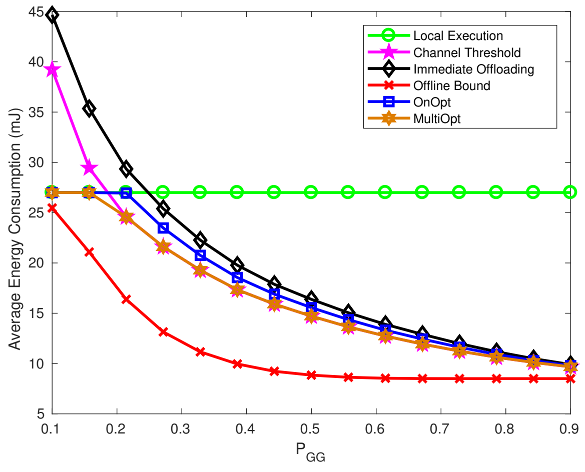

Fig. 4. Average energy consumption versus P_{GG}

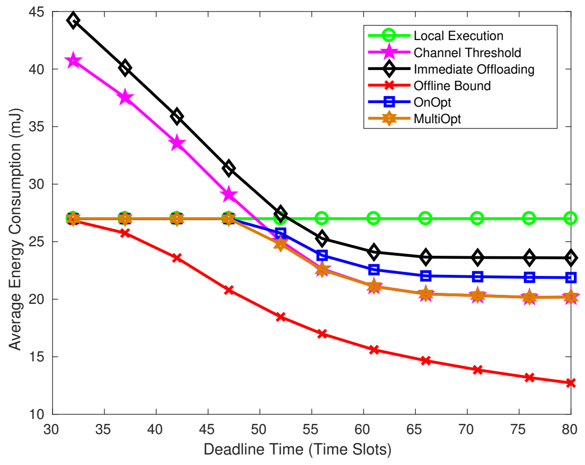

Fig. 5. Average energy consumption versus T_D: $P_{GG} = 0.3$

a continuous Good channel bit rate, which is not true during the actual uploading process, since it encounters more Bad channel states as S_{up} increases. Note that the average energy consumption of the offline bound is always the lowest, due to the future channel state information available to it. Compared to OnOpt, the energy consumption of MultiOpt is lower, as expected. By splitting the total amount of data into two parts, the MultiOpt algorithm has more flexibility that helps the mobile device to avoid uploading over long periods of bad channel states and save energy. This extra degree of freedom is not available to OnOpt, which has to continue uploading even over a bad channel once it has committed to offloading.

Next, we use the application parameters for x264 (H.264) encoding from [13], and consider a job with $S_{up} = 80$Kb, needing $D = 18$M CPU cycles, and $T_D = 80$ 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. The results are shown in Figures 4 and 5.

Figure 4 shows the average energy consumption of different algorithms as P_{GG} varies. When P_{GG} is small, channel condition is poor, the offline bound, MultiOpt, and OnOpt algorithms are more likely to decide to not offload, resulting in energy consumption very close to that of Local Execution, while the Immediate Offloading and Channel Threshold algorithms may consume much more energy than the latter by offloading. As P_{GG} increases, the average energy consumption of all the offloading algorithms decreases, since a shorter time is needed to complete the uploading due to better channel conditions. The energy consumption of the offline bound decreases with P_{GG} much faster than that of the other offloading algorithms due to available future information, and then becomes almost constant when P_{GG} is sufficiently large (e.g., exceeds 0.5 in Figure 4). Immediate Offloading results in the highest energy consumption among all the algorithms. As expected, the average energy consumption of the proposed MultiOpt algorithm is lower than that of the OnOpt for all P_{GG} values, and for the same reasons as above. When P_{GG} is close to 1, all offloading algorithms have about the same average energy consumption, since the channel conditions are almost always Good, and all the algorithms make the same offloading decisions.

Figure 5 shows the average energy consumption of the algorithms as the job deadline T_D changes. In general, as T_D increases, the average energy consumption of all the offloading algorithms decreases, while that of the Local Execution is not affected. When T_D is small, offloading is less likely to meet the deadline requirement; therefore, the offline bound, MultiOpt and OnOpt algorithms are more likely to decide not to offload, and, as a result, the average energy consumption of these algorithms is the same as or close to the energy consumption of Local Execution. When T_D is sufficiently large, the MultiOpt algorithm is almost the same as the Channel Threshold algorithm in terms of average energy consumption, since both algorithms end up deciding to offload at the earliest Good state for each part of the data.

VI. Conclusions

This paper considered the issue of multi-decision mobile computation offloading. Multi-decision offloading can be used to reduce mobile energy use by partitioning the upload of a task to be executed into separate parts, rather than offloading it as a single contiguous multi-packet upload. The paper considered the multi-decision problem when task execution completion times are subject to hard deadline constraints, and when the wireless channel can be modelled as a Markov process. An online mobile computation offloading algorithm, MultiOpt (Multi-decision online Optimum), was introduced, and was proven to be energy-optimal. Although the proposed algorithm is proven to be optimal, the paper also presented results using the Gilbert-Elliott channel model, which is commonly used to model burst noise effects. The proposed algorithm performance was compared to four others, namely, Immediate Offloading, Channel Threshold, Local Execution and optimal single-part offloading. Our simulation results show that the proposed algorithm can significantly improve mobile device energy consumption even when compared to energy optimal single part offloading, over a wide range of system parameters.

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] 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.

[3] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative Task Execution in Mobile Cloud Computing Under a Stochastic Wireless Channel," *IEEE Trans. on Wireless Comm.*, vol. 14, no. 1, pp. 81–93, January 2015.

[4] H. Lagar-Cavilla, N. Tolia, E. D. Lara, M. Satyanarayanan, and D. OHallaron, "Interactive resource-intensive applications made easy," in *ACM/IFIP/USENIX International Conf. Middleware*, 2007, pp. 143–163.

[5] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-Optimal Mobile Cloud Computing under Stochastic Wireless Channel," *IEEE Trans. on Wireless Comm.*, vol. 12, no. 9, pp. 4569–4581, September 2013.

[6] A. Hekmati, P. Teymoori, T. D. Todd, D. Zhao, and G. Karakostas, "Optimal mobile computation offloading with hard deadline constraints," 2018, submitted for publication.

[7] C. You, K. Huang, and H. Chae, "Energy Efficient Mobile Cloud Computing Powered by Wireless Energy Transfer," *IEEE Journal on Selected Areas in Comm.*, vol. 34, no. 5, pp. 1757–1771, May 2016.

[8] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, December 2016.

[9] C. M. Grinstead and J. L. Snell, "Chapter 11 - markov chains," in *Grinstead and Snells Introduction to Probability*, 2006, pp. 405–470.

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

[11] 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.

[12] 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.

[13] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," in *Proc. of the 2nd USENIX conference on hot topics in cloud computing, Berkeley, CA, USA*, 2010.