# Optimal Mobile Computation Offloading With Hard Deadline Constraints

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

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

***Abstract*—This paper considers mobile computation offloading where task completion times are subject to hard deadline constraints. Hard deadlines are difficult to meet in conventional computation offloading due to the stochastic nature of the wireless channels involved. Rather than using binary offload decisions, we permit concurrent remote and local job execution when it is needed to ensure task completion deadlines. The paper addresses this problem for homogeneous Markovian wireless channel models. An online energy-optimal computation offloading algorithm, OnOpt, is proposed. Its energy optimality is shown by constructing a time-dilated absorbing Markov process and applying dynamic programming. Closed form results are derived for general Markovian processes, and the Gilbert-Elliott channel model is used to show how the particular structure of the Markov chain can be exploited in computing optimal offload initiation times more efficiently. It is shown that job completion time probabilities can be computed recursively, which leads to a significant reduction in the computational complexity of OnOpt. The performance of the proposed algorithm is compared to three others, namely, Immediate Offloading, Channel Threshold, and Local Execution. Performance 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*—Cloud computing, mobile computation offloading, energy efficiency, mobile task execution performance, hard job deadline constraints.**

## I. Introduction

Mobile devices are continuing to become more pervasive as personal computing platforms. This trend is coinciding with significant increases in mobile application features that benefit from tight interactions with fixed computation infrastructure. According to a recent report, Cisco Inc. predicts that by the year 2021, monthly world-wide mobile data traffic will approach 28 exabytes [1]. Due to their limited physical size however, mobile devices are inherently resource-constrained, especially from an energy and computational viewpoint. The former constraint limits mobile battery lifetime [2], which is by far the most common smartphone complaint. This has motivated a wide variety of recent research on mobile energy efficiency [3].

Mobile cloud computing has been introduced to help alleviate some of these shortcomings, and to support the ever increasing computation and storage demands for mobile devices [4] [5]. It has been estimated that tens of billions of cloud-based network edge devices will be deployed in the future to satisfy mobile demands. This will provide significant resources for performing computation intensive and latency-critical mobile-centric tasks [6] [7]. Mobile computation offloading has been proposed as a way of decreasing mobile device energy use by dynamically offloading job execution to infrastructure based cloud servers [8] [9] [10] [11] [12]. It has been demonstrated that task offloading can significantly improve battery lifetime compared to the non-offloading case [13] [14].

Various architectures have been proposed for mobile computation offloading. Reference [15] originally proposed an architecture known as MAUI, which controls computation offloading for runtime .NET applications by formulating the offloading problem as a linear program. A similar architecture for Android applications has also been proposed in [8]. For a single user offloading its entire application to the cloud, the tradeoff between energy saving and computing performance was studied in [3] [16] [17]. References [18] [19] [20] [21] considered multi-user scenarios with a single application or user task, where the entire application is offloaded to the cloud. Unlike whole-application offloading. References [8] [15] [22] [23] and [24] have considered partitioning applications into multiple offloaded tasks. Mobile computation offloading has also been studied from a game theoretic viewpoint where resource contention may occur on the wireless channel or at the cloud servers [25] [26] [27] [28].

In this paper we study mobile computation offloading where job completion times are subjected to *hard* deadline constraints, i.e., deadline constraints that should *never* be violated, as opposed to deadline constraints that are satisfied with high probability, or incur a penalty when violated, etc. (cf. Section

II below). This objective will become increasingly important as mobile applications become more sophisticated and interact more closely with cloud job execution [29]. Hard deadlines, however, are often difficult to achieve in mobile networks due to the randomness of the wireless channels used for the mobile/cloud data interactions. In harsh wireless conditions, for example, complete channel outage can even occur over extended time periods. In this work, we study the straightforward approach of permitting *concurrent* local and cloud offload execution when a job completion deadline must be respected. This is in contrast to the conventional computation offloading model where job execution is either local or remote [8] [15]. As is the case in conventional computation offloading, the objective is to reduce the mobile device energy needed for job execution.

The paper studies this problem for Markovian wireless channel models [30] [31]. An online computation offloading algorithm, referred to as OnOpt (Online Optimum), is proposed. It is shown that OnOpt satisfies job deadlines and is optimum from a mean energy viewpoint. This is proven by augmenting the underlying channel model so that it forms a time-dilated absorbing Markov process. Dynamic programming is then used to establish a test that determines whether a given job should be offloaded at the current time, or to wait for some future offload opportunity. The performance results presented use the Gilbert-Elliott channel model. In this case, we exploit the structure of the channel Markov chain to compute the job completion time probabilities recursively, and this results in a significant reduction in the computational complexity of the proposed algorithm. The performance of the proposed algorithm is compared to three simpler heuristics, namely, offload immediately (Immediate Offloading), wait until the channel condition improves to above a threshold (Channel Threshold), and execute the job only locally (Local Execution). An offline lower bound on energy consumption is also computed and used for comparisons. Performance results show that the proposed algorithm can significantly improve mobile device energy consumption compared to the other approaches.

The main contributions of the paper are summarized, as follows.

1) To the best of our knowledge, this is the first work that uses computation offloading to reduce mobile energy and provides a mechanism for guaranteeing that hard job deadlines are always satisfied, even in the presence of full wireless channel outage conditions.
2) An online offloading decision algorithm, i.e., OnOpt (Online Optimal), is introduced. It is theoretically proven that the algorithm not only satisfies hard deadline constraints of the applications with certainty, but also achieves the minimum mean mobile device energy possible for homogeneous Markovian wireless channels.
3) An integer program (IP) is formulated that computes a strict lower bound on mobile device energy. This bound is used for comparisons in our performance results.
4) Closed form results are derived for obtaining job completion time probabilities for the homogeneous Markovian wireless channel case.
5) Although the proposed OnOpt algorithm satisfies hard deadlines and is proven to be energy optimal, performance results are also presented that compare it with the computation offloading heuristics: Immediate Offloading, Channel Threshold and Local Execution. These algorithms also ensure that hard job deadlines are preserved.

The rest of this paper is organized as follows. Section II discusses previous work that is most related to our paper. In Section III, we describe the system and present a model for local and remote job execution that satisfies hard job execution deadline constraints. Then, in Section IV, we derive an offline lower bound on the energy consumption, which is plotted in the results section and compared to various offloading algorithms. In Section V we discuss the Markovian channel model and how it is used to form a time-dilated absorbing Markov chain. This construction permits us to apply dynamic programming and come up with the energy optimum online algorithm OnOpt, proposed in Section VI. Then, in Section VII the paper focuses on the Gilbert-Elliot channel model, where it is shown that calculations can be performed efficiently, decreasing the complexity of OnOpt. In Section VIII, performance results are presented that compare OnOpt with various other computation offloading algorithms that ensure that hard job deadlines are preserved. Finally, we present our conclusions in Section IX.

## II. Related Work

Many mobile cloud computing issues and challenges have been addressed in the past few years [32] [33] [34] [35]. A significant part of the literature has considered mobile computation offloading issues under stochastic transmission channel and cloud server conditions. Reference [3], for example, presents an energy model to analyze offloading, mainly considering mobile computation and communication energy components based on statistical inputs and with fixed wireless channel conditions. This work analyzed the offloading policy assuming that network conditions remain static throughout the offloading/execution process. Network prediction was used as inputs to the decision process. In [36] a method was proposed for energy-optimal mobile cloud computing under stochastic wireless channels. The issue of job deadlines was considered from a statistical viewpoint, rather than enforcing hard job execution deadlines. Dynamic programming was used in [37] to optimize offloading decisions from an energy viewpoint, but the issue of job execution time constraints was not considered. In Reference [29], job execution time constraints were flagged as a key issue for many interactive applications. The difficulties of achieving this under random channel conditions was highlighted. In [38], a framework was proposed for executing jobs either locally, by CPU frequency scheduling, or remotely, by offloading over a stochastic channel. In the latter case, mobile transmit power control is used to select bit rates to ensure that job deadlines are met. In local execution, a violation parameter is defined that permits the execution to probabilistically exceed the deadline, and, therefore, the latter is not "hard" in our sense. As in our paper,

this work uses the well-known Gilbert-Elliot channel model for its results. Computation offloading for 3D video streaming was studied in [39], where a cloud-enabled smart camera network allows smart cameras to continuously offload computationally intensive tasks to a remote cloud server, which helps the smart cameras with the video encoding and decoding. Experiments show that using this system can extend the battery life of the smart cameras without compromising compression efficiency or video quality. The system considered in [40] consists of multiple brokers that coordinate cloud (including a public cloud and a cloudlet) resource allocations among mobile users. Each broker minimizes the average price for its users to use the cloud services, while satisfying predefined quality of experience constraints of the users. The price for a mobile user is a monetary cost that is proportional to the amount of public cloud resources allocated to the user or determined by the bidding strategy when competing for the cloudlet resources. Reference [41] addressed the problem of computation offloading on multicore-based mobile devices running multiple applications, each of which consists of multiple mutually dependent tasks. In addition to determining whether a task should be performed locally or offloaded to the cloud, the work also considers executing local tasks on which CPU cores in order to optimize the energy consumption of the mobile devices. A static wireless channel with fixed bitrate is considered.

## III. 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. Each job could be a sub-task associated with multiple local/remote job execution components [4] [5]. We focus on a single task whose characteristics are known at its release time. Figure 1 defines the parameters used for job computation timing. Note that time is taken to be discrete, i.e., quantized into equal length *time slots* whose duration is normalized to 1. Time values are therefore referred to by their time slot indices. Note that the time slot duration is defined to accommodate the channel propagation model discussed in Section V 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.
- $S_{down}$:Number of bits transmitted through the downlink channel when downloading job results from the cloud.

We now discuss the timing and energy use associated with local and remote offloaded job execution.

### A. Local Execution

It is assumed that the energy cost and time needed to execute a job locally is known at the job release time, $t_r$, and these are defined by $E_L$ and $T_L$, respectively. While this may not always be the case, this assumption is often true and has been made in many computational offloading studies [3], [19], [28].

If the computation offloading algorithm elects to execute the job locally without any remote offloading, we must ensure that the job deadline is always satisfied. Therefore, local execution must start no later than

$$t_L = t_D - T_L + 1, \tag{1}$$

unless remote offload/execution results are available at the mobile device before $t_L$, i.e., local execution must start $T_L$ time slots prior to the job deadline, if remote execution results have not arrived by then. This is shown in Figure 1.

Note that starting the local job execution at time slot $t_L$ ensures the hard delay constraint of the task, if a remote offloading response is not received in time. Although this may result in both local and remote executions of the task, it will always satisfy the hard deadline, even if there is channel contention or extended channel outages. However, with the objective of minimizing the mean energy consumption of the mobile device, the proposed algorithm will reduce the possibility of both local and remote executions.

### B. Remote Execution

In the case of offloading a job, we will assume that, upon its release, the job is assigned an execution time $T_{exec}$ by the cloud server, which is communicated to the mobile device (or is prescribed by, say, the contractual agreement between the user of the device and the cloud server operator). In addition, we assume that the user has been allocated capacity (such as recurring time slots) until the offload has completed. These assumptions are commonly invoked [3] [19] [28]. Therefore, if $T_{up}$ and $T_{down}$ are the time periods needed to, upload the job to the cloud server, and, download its results to the device, respectively, the total offloading time $T_{off}$ is given by

$$T_{off} = T_{up} + T_{exec} + T_{down}. \tag{2}$$

These components are shown in Figure 1, where we have defined $t_o$ to be the remote offload initiation time. It is assumed that the channel uses bit rate adaptation to accommodate random variations in channel conditions. As a result, $T_{up}$ is a random variable, dependent on the evolution of the uplink channel state as a given upload occurs. In what follows, it is assumed that the channel state can be modelled as a homogeneous discrete-time Markov process; the same holds for $T_{down}$.

In order to simplify our exposition, we will initially focus on the randomness induced by the Markovian uplink channel. In the following development, we therefore temporarily assume that all offloading deadlines, job sizes (in bits), and energy costs are related only to job uploading, i.e., $T_{off} \equiv T_{up}$

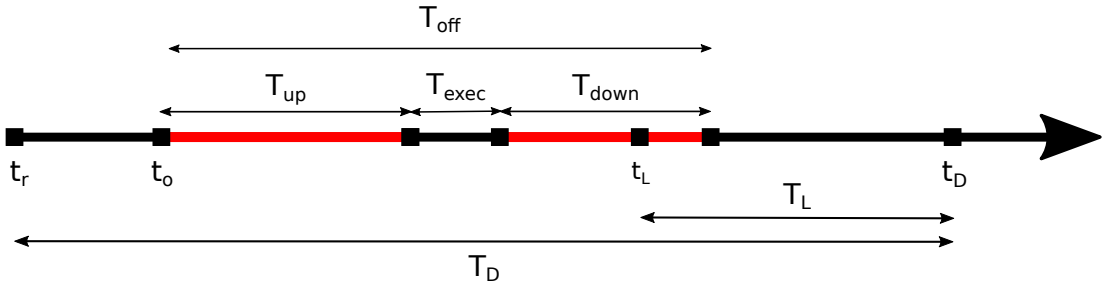


Fig. 1. Job Computation Timing. The job release time is $t_r$ and its deadline is $t_D$. The offload begins at $t_o$ and execution is completed $T_{up}+T_{exec}+T_{down}$ time slots later. To enforce the job deadline, local execution must begin at $t_L$ if the mobile is still awaiting a remote response. At time $t_o+T_{up}+T_{exec}+T_{down}$, local execution is terminated provided that a remote offload response arrives before $t_D$. Note that by definition, when $t_o > t_D$, then there is only local execution.

and $S \equiv S_{up}$, so that $T_{exec} = T_{down} = 0$. Given the ensuing results, adding the effects of $T_{exec}$ and $T_{down}$ is straightforward and is deferred to Section VII. At that point we discuss, for example, how we include the effects of a random Markovian downlink channel.

Since the job's hard deadline constraint must always be satisfied, we propose its simultaneous cloud server offloading (if possible and beneficial) *and* its local execution. Given the stochastic nature of the transmission channel, deciding whether and when to offload (i.e., $t_o$ in Figure 1), depends on the estimation of offloading energy consumption *and* offloading time, in order to both minimize energy costs for the mobile device, *and* satisfy the job deadline constraint.[1] Depending on these estimates, there are three possibilities for offloading at time slot $t_o$: (i) it certainly finishes before starting the local execution of the job, and, hence, local execution never starts, or, (ii) it finishes after starting the local execution of the job, and, possibly, *before* deadline $t_D$; then, the fraction of local execution energy cost incurred is equal to the fraction of $T_L$ overlapping with the offloading (i.e., local execution is terminated if a remote offload response is received), or (iii) it certainly finishes *after* deadline $t_D$, so it does not even start, and the total energy cost is equal to the local execution energy cost. Note that in the case of a deterministic channel, one can calculate exactly in which of these three cases the job falls. In this work, we analyze the problem of offloading with hard deadlines over a Markovian stochastic channel, described in detail in Section V.

As in most of the related work 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.

## IV. Offline bound

In this section, an offline lower bound on mobile device energy is derived. This bound is used in Section VIII 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. When a job is released, the bound then chooses the job offload time so that its deadline is met and the energy needed is minimized. Let $t_o$ be the time to start offloading, given that we know the bit rate $B_t$ (in bits per time slot) at all times $1 \le t \le t_D$ (recall that $t_r$ is taken to be 1). Let $t_f(t_o)$ be defined as the offload finishing time when offloading starts at $t_o$. Then $t_o$ can be found by solving the following IP.

$$\min_{t_o} \quad \frac{\max(t_o, t_L) - t_L}{T_L} E_L + \sum_{t=t_o}^{t_f(t_o)} e_t \tag{3}$$

$$s.t. \quad \frac{\max(t_o, t_L) - t_L}{T_L} E_L + \sum_{t=t_o}^{t_f(t_o)} e_t \le E_L \tag{4}$$

$$1 \le t_o \le t_D. \tag{5}$$

Objective (3) consists of two terms. The first is the local execution energy cost incurred before offloading starts. If $t_o < t_L$, this term is zero, which means that there has been no local execution to that point; otherwise, $\frac{t_o - t_L}{T_L} E_L$ is the energy that has been expended by local execution energy before $t_o$. The second term in (3) is the total energy consumption after offloading starts where $e_t$ is the energy expended in time slot $t$. When $t_o < t < t_L$, each $e_t$ includes only the offloading energy; and when $t \ge t_L$, both offloading and local execution are performed at time slot $t$. Therefore, $e_t$ is given as

$$e_t = \begin{cases} E_{tr}, & t < t_L \\ E_{tr} + \frac{E_L}{T_L}, & t \ge t_L \end{cases} \tag{6}$$

where $E_{tr}$ is the energy cost per time slot for transmitting on the channel. 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 is no feasible offloading start time $t_o$, i.e., it is best to execute locally without offloading.

## V. Markovian Channel and the Time-Dilated Absorbing Markov Model

In many studies, homogeneous Markov chains have been used to model random wireless channel conditions and as is often assumed, the Markovian transition probabilities are taken to be known, or have been learned dynamically [30] [31] [36] [38] [42] [43]. Accordingly, we assume that the computation offloading occurs over a finite state Markovian channel. In this case, the OnOpt (Online Optimal) algorithm proposed

[1] Note that when offloading occurs, then $t_r \le t_o \le t_D$, and when $t_o > t_D$, then there has been no offloading, i.e., there is only local job execution.

in Section VI is an online computation offloading algorithm that attains the minimum expected execution energy. As is commonly assumed, the channel data rate is defined by the Markovian channel state and the receive signal-to-noise ratio (SNR) is such that errors due to random noise are negligible. When this is not the case, then the execution time constraint will still be satisfied by the OnOpt algorithm [36] [43].

In this section we use the conventional channel state Markov chain (CSMC) to form a time-dilated absorbing Markov chain (TDAMC), which models the offloading over the channel. The resulting Markov process is used by OnOpt in order to compute its energy and offloading time estimates, and by our analysis, in order to show its optimality. As mentioned above, we focus on $T_{up}$, ignoring $T_{exec}$ and $T_{down}$ (cf. Figure 1); hence, $T_{off}$ and $S$ below refer to $T_{up}$ and $S_{up}$, respectively. Section VII-A below describes the straight-forward inclusion of $T_{exec}$ and $T_{down}$.

In the CSMC, and starting from the current time slot $t_s$, the channel conditions will evolve from one time slot to the next according to a homogeneous finite state Markov chain. We denote the set of possible channel states by $\mathcal{M}$, where $M = |\mathcal{M}|$ is the number of states in the CSMC. As discussed previously, the radio transmit power is fixed and bit rate adaptation is used to adjust to varying channel conditions. Therefore, each state in the CSMC has an associated bit rate that gives the number of bits per time slot that can be uploaded when offloading occurs in that state. In a general Markov chain model, 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 $i$. Unfortunately, CSMC is memoryless as far as the state of offloading and channel conditions are concerned; in order to incorporate them into our model, we form a new Markov chain, referred to as a *time-dilated absorbing Markov chain (TDAMC)*. We are again interested in the evolution of the system starting at the current time slot $t_s$, and running until the computation has completed, either locally or via offloading. The state of the channel in each TDAMC state at time $t \geq t_s$ is represented by $X_t$ where $X_t \in \mathcal{M}$. However, unlike the CSMC, the TDAMC incorporates $t$ and other information into its structure.

The TDAMC models the job offloading progress if the latter is initiated at the current time slot $t_s$. It is a rooted tree, constructed as follows: The root state is the channel state $X_{t_s}$ at current time slot $t_s$; since this is the current time slot, $X_{t_s}$ is known. 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 CSMC states. At each step along a given tree branch, the number of job bits transmitted is determined by the bit rate associated with the channel state in question. This construction continues along each branching tree path until the number of bits offloaded reaches the job upload size, $S = S_{up}$. At that point, the state reached in the TDAMC is defined as a Markov chain *absorbing* state, i.e., it has a self-transition with probability 1. From this construction it can be seen that the TDAMC includes all possible paths that lead to a successful job offload, and that all of the states are either transient or absorbing. Eventually, all paths terminate in an absorbing state, and the energy cost of that path is proportional to its length, i.e., the number of time slots needed.

An example of a TDAMC is shown in Figure 2, for $t_s = 1$. It is constructed from a two-state Gilbert-Elliot channel, which is modelled by a CSMC with $\mathcal{M} = \{G, B\}$ (i.e., with "Good" and "Bad" states, respectively), and transition probabilities matrix

$$\begin{bmatrix} P_{GG} & P_{GB} \\ P_{BG} & P_{BB} \end{bmatrix},$$

i.e., $P_{1,1} = P_{GG}, P_{1,2} = P_{GB}, P_{2,1} = P_{BG}$ and $P_{2,2} = P_{BB}$. In each time slot, the TDAMC transitions to a new state in accordance with these transition probabilities. For clarity, each channel state in the figure is subscripted with its level time and the index of the subtree it belongs to. For example, $G_{3,2}$ indicates that the channel state at level $t = 3$ and subtree 2 is Good. The TDAMC shows that at $t = 3$, the channel can remain in the $G$ state, i.e., $G_{4,2}$ or transition to the $B$ state, i.e., $B_{4,2}$ with the given CSMC transition probabilities. Each state of the TDAMC defines the number of bits that can be offloaded during a time slot while in that state. In the example of Figure 2, when the channel state is $G$, the number of payload bits is defined by the number of bits that can be carried on the channel during a good (i.e., high bit-rate) channel state. In the general case, when the channel is in state $X_t$ at time $t$, the number of child states at $t+1$ is given by the number of non-zero values in the same row of the original CSMC transition matrix. In Figure 2, each state continues to branch downwards until the number of offloaded bits for a given branch reaches the total number needed for the offload. At that point, the branch ends in a Markov chain absorbing state discussed previously. In Figure 2, states $G_{3,1}$, $G_{4,1}$ and $G_{4,2}$ are absorbing states.

The non-absorbing states in the TDAMC are clearly all transient states. We define $\mathcal{A}$ to be the number of absorbing states and $\mathcal{T}$ to be the number of transient states in the TDAMC. For an absorbing Markov chain, by labeling the transient states first, the resulting transition matrix can be written in the following form [44]:

$$\mathbb{P}_{\text{TDAMC}} = \begin{bmatrix} Q & R \\ \mathbf{0} & I_{\mathcal{A}} \end{bmatrix}. \tag{7}$$

In $\mathbb{P}_{\text{TDAMC}}$, the $\mathcal{T} \times \mathcal{T}$ sub-matrix $Q$ contains the probabilities of transitioning between transient states before the job upload is completed. The $\mathcal{T} \times \mathcal{A}$ sub-matrix $R$ contains the probabilities of transitioning from a transient state to an absorbing state, indicating that the job upload is finished. $\mathbf{0}$ is an $\mathcal{A} \times \mathcal{T}$ zero matrix and $I_{\mathcal{A}}$ is an $\mathcal{A} \times \mathcal{A}$ identity (i.e., absorbing) matrix.

$Q$ contains the entries of the original CSMC transition matrix that give the transition probabilities of each state $k$ when it transits to a state in $\{s_k, s_k+1, \cdots, f_k\}$, and, for our TDAMC, it has the following form:

$$Q = \begin{bmatrix} 0 & P_{1,s_1} & \cdots & P_{1,f_1} & 0 & \cdots & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & P_{2,s_2} & \cdots & P_{2,f_2} & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}.$$

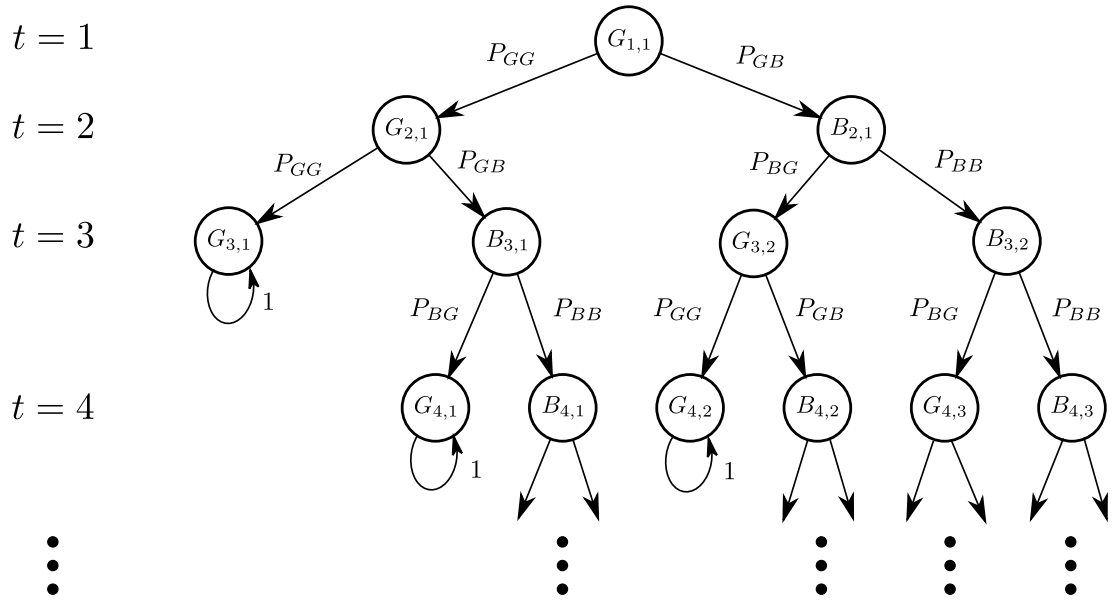


Fig. 2. Time Dilated Absorbing Markov Chain Example: This example corresponds to that obtained from an underlying Gilbert-Elliot channel model. In each time slot, the Markov chain transitions to a new state in accordance with the transition probabilities from the Gilbert-Elliot model.

It can be seen that $Q$ is upper triangular, as expected, since all states are transient and can be visited at most once. The (possibly) non-zero transition probabilities shown in row one, for example, give the probability of transitioning to all possible $t = 2$ channel states and so on.

With the above construction and using results from the theory of absorbing Markov chains, various statistics can be computed by first forming the fundamental matrix

$$N = (I - Q)^{-1}. \tag{8}$$

For example, entry $(i, j)$ of $N$ gives the expected number of times that the TDAMC is in transient state $j$ if the system is started in transient state $i$.

Due to the structure of our TDAMC, the computation needed in Equation (8) can be greatly simplified. Note that $N^{-1}$ is still an upper triangular matrix with all the diagonal entries equal to one, and can be decomposed as follows:

$$N^{-1} = N_{\mathcal{T}} N_{\mathcal{T}-1} N_{\mathcal{T}-2} \cdots N_1,$$

where

$$N_k = \begin{bmatrix} 1 & 0 & \cdots & 0 & n_{1,k} & \cdots & 0 \\ 0 & 1 & \cdots & 0 & n_{2,k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & n_{k-1,k} & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

$N_k$ is an atomic triangular matrix whose inverse is given by

$$N_k^{-1} = \begin{bmatrix} 1 & 0 & \cdots & 0 & -n_{1,k} & \cdots & 0 \\ 0 & 1 & \cdots & 0 & -n_{2,k} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & -n_{k-1,k} & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Then

$$N = (N_{\mathcal{T}} N_{\mathcal{T}-1} N_{\mathcal{T}-2} \cdots N_1)^{-1} = N_1^{-1} N_2^{-1} N_3^{-1} \cdots N_{\mathcal{T}}^{-1}$$

Note that each column of the $Q$ matrix has only one nonzero element. Therefore, $N^{-1}$ will have only two nonzero elements in each column. Similarly, in $N_k$ only one of the $n_{1,k}, n_{2,k}, \ldots, n_{k-1,k}$ is non-zero. Therefore, the multiplication can be done efficiently.

The absorption probabilities for all absorbing states can be obtained by

$$W = NR, \tag{9}$$

where $W$ is a $\mathcal{T} \times \mathcal{A}$ matrix and $W[i, j]$ gives the probability that a particular absorbing state $j$ will be reached if the system starts in transient state $i$. Using this procedure, we can thus compute the various probabilities of absorption for each absorbing state, given knowledge of the starting state. Therefore, we can obtain the probability of finishing the offload for every possible offloading time $T_{off}$ by summing all of the absorbing state probabilities that have the same TDAMC path length. We define $P_t(T, x)$ to be the probability of offloading in exactly $T$ time slots, when offloading starts at time $t$ with the channel in state $X_t = x$. Then

$$P_t(T_{off}, x) = \sum_{j \in \mathcal{S}} W[x, j] \tag{10}$$

where $\mathcal{S}$ are all of the entries of the matrix where the offloading time is equal to $T_{off}$. Note that $P_t(T, x) = 0$ when it is impossible to offload in a period of exactly $T$ time slots when offloading at $t$ with the channel in state $X_t = x$, i.e, $T$ is shorter (longer) than the shortest (longest) time needed to offload, under the best (worst) channel conditions. $P_t(T, x)$ is critical for computing the expected cost of offloading used by the algorithm OnOpt (cf. Section VI).

The ability to compute the $P_t(T, x)$ values allows for the computation of the energy costs for both offloading and local execution. If offloading the job (of bit size $S$) starts at time slot $t$, its expected transmission energy is calculated as follows, depending on whether

- offloading is certainly completed ($1 \leq t < t_D - \frac{S}{B_{min}} + 1$), in which case the energy spent is proportional to $T_{off}$.
- offloading may or may not be completed within the deadline ($t_D - \frac{S}{B_{min}} + 1 \leq t \leq t_D$), in which case the energy cost is $T_{off}$ or $t_D - t + 1$, respectively (clearly the deadline $t_D$ is the last time slot where offloading can be done).

Noting that $P_t(T_{off}, x) = 0$ when $T_{off} < \frac{S}{B_{max}}$ or $T_{off} > \frac{S}{B_{min}}$,[2] the expected offloading energy cost when offloading starts at time slot $t$ with the channel in state $x$, is given by (11).

Recall that local execution is postponed until the very last moment, i.e., time slot $t_L = t_D - T_L + 1$, where $T_L$ is the number of time slots needed by the task to execute locally. A central idea of this paper is that, although local execution is always initiated (if offloading has not completed earlier) at time $t_L$, in order to guarantee completion within the deadline, offloading will be decided in such a way so that it will (hopefully) terminate *before* $t_D$, thus saving us the energy cost of the remaining local execution. The overlap time (when such exists) between offloading at time $t$ and local execution is $\min\{t_D + 1, t + T_{off}\} - t_L$. By recalling that $E_L$ is the energy cost of complete local execution of the task, the local execution energy cost will be 0 if there is no overlap, or a fraction $\frac{\min\{t_D+1, t+T_{off}\}-t_L}{T_L}$ of $E_L$ if there is. Hence, we obtain that the expected local execution cost when offloading starts at time $t$ with the channel in state $x$, is given by (12). In the first case, there will be overlap only for $T_{off} \geq t_L - t + 1$, while in the second there is always overlap, since $t - t_L + T_{off} > 0$.

## VI. Optimal Offloading Starting Time and the OnOpt (Online Optimal) Algorithm

In this section we use the time-dilated absorbing Markov model construction of Section V and the theory of optimal stopping for Markov decision processes [45] to define the OnOpt algorithm, and show that it achieves the optimal mean energy for the mobile device. A high-level description of the algorithm is as follows: At each time slot $t$ (starting from the job release time slot), the algorithm considers the TDAMC model for starting offloading at current time $t_s = t$. It computes (based on the TDAMC) the optimal offloading starting time $\tau^*_{t_D} \geq t$, by formulating the problem as a Markovian optimal stopping problem. If $\tau^*_{t_D} = t$, then offloading is started immediately at time $t$. Otherwise, the algorithm waits till time slot $t + 1$, to repeat the above process.

Suppose that the current time slot is $t_s$, and consider the corresponding TDAMC rooted at state $X_{t_s}$. In order to compute the optimal time slot for starting offloading (if offloading turns out to be more beneficial, in expectation, than executing the task solely locally), we need to compute the *offloading starting time* $\tau^*_{t_D}$ that satisfies the following optimization problem:

$$v_{t_D}(y) = \min_{t: t_s \leq t \leq t_D+1} E[g_t(X_t)|X_{t_s} = y] \tag{13}$$

$$= \min_{t: t_s \leq t \leq t_D+1} \sum_{z \in \mathcal{M}} Pr[X_t = z|X_{t_s} = y] g_t(z), \tag{14}$$

where $X_{t_s}$ is the current channel state, and $g_t(x)$ is the expected total energy cost if offloading starts at time slot $t$ with channel state $X_t = x$. The choice of $t = t_D + 1$ in (13) corresponds to no offloading, in which case (11) and (12) imply a total cost of $E_L$. Then, for $t_s \leq t \leq t_D$,

$$g_t(x) = E_{off}(t, x) + E_L(t, x), \tag{15}$$

where $E_{off}(t, x), E_L(t, x)$ are the expected offloading and local execution costs, respectively, as defined in (11) and (12), when offloading starts at time $t$ with the channel in state $X_t = x$.

The optimization problem (13) is inherently an off-line problem, while the algorithm we would like to use is inherently an on-line one, in the sense that at every time slot it has to decide whether to offload or not, *given the history of channel states it has encountered so far.* Such an algorithm is defined by the following recursion, which can be solved using Dynamic Programming (DP), i.e.,

$$V_t(x) = \begin{cases} E_L, & t \geq t_D \\ \min\{g_t(x), E[V_{t+1}|X_t = x]\}, & t = t_s, \cdots, t_D - 1 \end{cases} \tag{16}$$

Note that $V_t(x)$ is the minimum between the expected total cost of offloading at the current time slot $t$, and the expected cost of postponing that decision to time slot $t + 1$, given that the channel state at time $t$ is $x$, and $E[V_{t+1}|X_t = x]$ is the expectation of $V_{t+1}(X_{t+1})$ over all possible $X_{t+1}$, under the condition that $X_t = x$, i.e.,

$$E[V_{t+1}|X_t = x] = \sum_{y \in \mathcal{M}} Pr[X_{t+1} = y|X_t = x] V_{t+1}(y).$$

Note that (16) implies a *policy*, that dictates whether at any time $t$ and state $X_t$ the algorithm should start uploading (if the $\min$ is attained by $g_t$), or should otherwise wait.

It is well known (e.g., Theorem 1.7 in [45]) that policy $V_t$ in (16) is *optimal*, i.e., it solves the original problem (13), since

$$v_{t_D}(y) = V_{t_s}(y), \ \forall y. \tag{17}$$

Therefore, the following lemma holds:

[2] We will assume that $\frac{S}{B_{max}}$ and $\frac{S}{B_{min}}$ are integers, to avoid burdening our formulas with ceilings $\left\lceil \frac{S}{B_{max}} \right\rceil$ and $\left\lceil \frac{S}{B_{min}} \right\rceil$.

$$E_{off}(t,x)=\begin{cases} E_{tr}\sum\limits_{T_{off}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_t(T_{off},x)T_{off}, & 1\le t<t_D-\frac{S}{B_{min}}+1 \\ E_{tr}\left(\sum\limits_{T_{off}=\frac{S}{B_{max}}}^{t_D-t} P_t(T_{off},x)T_{off}+\sum\limits_{T_{off}=t_D-t+1}^{\frac{S}{B_{min}}} P_t(T_{off},x)(t_D-t+1)\right), & t_D-\frac{S}{B_{min}}+1\le t\le t_D \\ 0 & t>t_D \end{cases} \tag{11}$$

$$E_L(t,x)=\begin{cases} \sum\limits_{T_{off}=t_L-t+1}^{\frac{S}{B_{min}}} P_t(T_{off},x)\left(\frac{\min\{t_D+1,t+T_{off}\}-t_L}{T_L}E_L\right), & 1\le t<t_L \\ \sum\limits_{T_{off}=\frac{S}{B_{max}}}^{\frac{S}{B_{min}}} P_t(T_{off},x)\left(\frac{\min\{t_D+1,t+T_{off}\}-t_L}{T_L}E_L\right), & t_L\le t\le t_D \\ E_L & t>t_D \end{cases} \tag{12}$$

**Lemma 1. [45]** *The optimal offloading starting time for* (13) *is* $\tau^*_{t_D}=\arg\min_{t_s\le t\le t_D+1}\{V_t(x)=g_t(x)\}$.

Lemma 1 implies that the on-line algorithm OnOpt, given in Algorithm 1, is optimal. Note that this result is true for any Markovian channel. The algorithm is given the local execution starting time $t_L$, local execution energy $E_L$, job deadline $t_D$, and job size $S$. It then arranges for the job to be executed either locally or by remote offloading (or both, if needed). Initially, the remote offload is disabled by setting $t_o$ to a value greater than $t_D$ in Line 1. At each time slot $t_s$ with the channel at state $X_{t_s}=x$, we test if $t_s<t_o$, i.e., no offload has been initiated for the job. Then both $g_{t_s}(x)$ and $E[V_{t_s+1}|X_{t_s}=x]$ are computed (using (15) and using DP to solve (16), respectively). If $g_{t_s}(x)\le E[V_{t_s+1}|X_{t_s}=x]$, then the offload begins at time $t_s$, i.e., $t_o=t_s$, since in this case $\tau^*_{t_D}=t_s$ from Lemma 1. If offloading finishes before a local execution finishes, then local execution is terminated (Line 11). At Line 13 we check to see if local execution should start so that the job's deadline can be guaranteed. Similarly, Line 16 tests if the local job has completed. In that case, any remote offload in progress will be terminated.

**Algorithm 1** OnOpt (Online Optimal) Algorithm

**Input:** Local execution starting time $t_L$, local execution energy $E_L$, job deadline $t_D$, and job size $S$.

1: $t_o:=\infty$ ▷ Offloading initially disabled ($t_o$ is offload start time)
2: **for all** $t_s\in\{1,\dots,t_D\}$ **do**
3: **if** $t_s<t_o$ **then**
4: $\overline{c_{t_s}}:=g_{t_s}(x)$ ▷ Expected energy cost of offloading at $t_s$.
5: $\overline{c_{t_s+1}}:=E[V_{t_s+1}|X_{t_s}=x]$ ▷ Expected energy cost of waiting until $t_s+1$.
6: **if** $\overline{c_{t_s}}\le\overline{c_{t_s+1}}$ **then**
7: $t_o:=t_s$ ▷ Start offloading.
8: **end if**
9: **else if** offloading terminates at $t_s$ **then**
10: Abort local execution (if active). ▷ Remote offload response has been received.
11: **return**
12: **end if**
13: **if** $t_s=t_D-T_L+1$ **then**
14: Start local execution. ▷ Ensure that the job deadline is satisfied.
15: **end if**
16: **if** $t_s=t_D$ **then**
17: Abort remote offload (if active). ▷ Local execution has completed.
18: **return**
19: **end if**
20: **end for**

## VII. The Gilbert-Elliot Channel Case

In this section, we consider the well-known Gilbert-Elliot channel model [30] [31], which has been used in many studies to model stochastic communication channels, e.g., [36] [38] [42] [43] [46], and will be used in the results section of this paper. This channel model is typically used to characterize the effects of burst noise in wireless channels, i.e., where the channel can abruptly transition from good to bad conditions (and vice versa). This type of channels is a difficult one for computation offloading algorithms to deal with, compared to one where there is much more correlation in the channel quality as the offloading progresses. In this section, closed form results are derived for this channel model that will be used to generate numerical results in Section VIII. With the two state channel model, we have $B_{max}=B_g$ and $B_{min}=B_b$, where $B_g$ and $B_b$ are the bit rates of the good and bad channel states, respectively (in bits per time slot). In order to run Algorithm 1 with the specific energy costs of (11) and (12), we need to calculate the probabilities $P_t(T_{off},X_t)$, which is the probability of an offload finishing in $T_{off}$ time slots, if it starts at time slot $t$ with channel state $X_t$.

Let $b$ be the number of bad state time slots during the $T_{off}$ offloading time slots. Given the data size $S$ to be offloaded, $b$

and $T_{off}$ must satisfy $S \leq bB_b + (T_{off} - b)B_g < S + B_g$. The upper bound is due to the fact that we transmit at most $S + B_g$ bits (we assume that even when the transmission of the useful $S$ bits has been completed, paying the transmission cost continues until the end of the last time slot). This implies that

$$\frac{(T_{off} - 1)B_g - S}{B_g - B_b} < b \leq \frac{T_{off}B_g - S}{B_g - B_b} \tag{18}$$

Define $\mathcal{B}$ as a set of integers $b$ satisfying (18). For any $b \in \mathcal{B}$, the actual transmitted number of bits, $\hat{S}$, is given by

$$\hat{S} = bB_b + (T_{off} - b)B_g. \tag{19}$$

Define $\hat{P}_t(T_{off}, b, X_t)$ as the probability of an offloading, that starts at time slot $t$ with state $X_t$ and takes $T_{off}$ time slots (among which $b$ time slots are in the bad states). We have that

$$P_t(T_{off}, X_t) = \sum_{b \in \mathcal{B}} \hat{P}_t(T_{off}, b, X_t). \tag{20}$$

Thus, $P_t(T_{off}, X_t)$ can be obtained by summing over all of possible $b$'s in $\hat{P}_t(T_{off}, b, X_t)$. As a special case, we set $\hat{P}_t(T_{off}, b, X_t) = 0$ for all $T_{off}$ and $X_t$ when $b < 0$. In order to derive $\hat{P}_t(T_{off}, b, X_t)$, we need the following lemma.

**Lemma 2.** *If $\hat{S} - S \geq B_b$, then the final transmission state must be G.*

*Proof:* Assume, for contradiction, that the final state is $B$. Then, the number of bits transmitted in $T_{off} - 1$ time slots is $\hat{S}_{T_{off}-1} \geq \hat{S}_{T_{off}} - B_b$. Given the condition of the lemma, this implies that $\hat{S}_{T_{off}-1} - S \geq 0$, i.e., offloading finished within $T_{off} - 1$ time slots, a contradiction. ∎

Based on Lemma 2 and $X_t$, four different cases are considered when calculating $\hat{P}_t(T_{off}, b, X_t)$, and are obtained from elementary counting:

- $X_t = G$ and $\hat{S} - S \geq B_b$: See (21).
- $X_t = G$ and $\hat{S} - S < B_b$: See (22).
- $X_t = B$ and $\hat{S} - S \geq B_b$: See (23).
- $X_t = B$ and $\hat{S} - S < B_b$: See (24).

Although equations (21)-(24) can be used to calculate $P_t(T_{off}, X_t)$ directly, we now show how they can be computed recursively, which leads to a significant reduction in computation time (albeit with the use of more memory). We show this for the case $\hat{S} - S \geq B_b$ and $X_t$ is Good (the other cases are handled similarly). In that case, (21) applies. We assume $b > 0$ (case $b = 0$ is trivial). Then, (21) for $b > 0$ implies

$$\hat{P}_t(T_{off}, b, Good) = \sum_{k=0}^{\min\{b-1, T_{off}-b-2\}} Z(k, T_{off}) \tag{25}$$

where

$$Z(k, T_{off}) = \binom{b-1}{k} \binom{T_{off} - b - 1}{k+1} P_{GB}^{k+1} \, P_{BG}^{k+1} \, P_{BB}^{b-k-1} \, P_{GG}^{T_{off}-b-k-2} \tag{26}$$

and

$$Z(0, T_{off}) = (T_{off} - b - 1) P_{GB} \, P_{BG} \, P_{BB}^{b-1} \, P_{GG}^{T_{off}-b-2} \tag{27}$$

Then, it is easy to see that

$$Z(k+1, T_{off}) = \frac{(b-k-1)(T_{off} - b - k - 2)}{(k+1)(k+2)} \frac{P_{GB}P_{BG}}{P_{BB}P_{GG}} Z(k, T_{off}) \tag{28}$$

for all $0 \leq k \leq \frac{S}{B_b}$. By treating $B_b$ and $B_g$ as constant, precomputing $Z(k, T_{off})$ for all $0 \leq k \leq \frac{S}{B_b}$ and $\frac{S}{B_g} \leq T_{off} \leq \frac{S}{B_b}$ takes $O(S^2)$ operations when (27) and (28) are used. Then, for any value of $T_{off}$, each $\hat{P}_t(T_{off}, b, G)$ can be computed with $O(S)$ operations from (25); eventually, $O(1)$ $\hat{P}_t$ values are combined to compute each $P_t(T_{off}, G)$ from (20) (note that $|\mathcal{B}| = O(1)$, and that $\hat{P}_t, P_t$ do not depend on $t$, except for defining $X_t$ in their arguments). Hence, we can precompute (and store) all possible $P_t(T_{off}, X_t)$ using $O(S^2)$ operations (and memory) overall. After that, (11) and (12) imply that we can calculate $E_{off}(t, x)$ and $E_L(t, x)$ for each $1 \leq t \leq t_D$ with $O(S)$ arithmetic operations. This implies that we can use (15) to precompute (and store) all $g_t(x)$'s using $O(ST_D)$ operations (and memory) overall, and, therefore, all $V_t(x)$'s using $O(ST_D)$ operations (and memory), using the recursive definition (16). After this $O(S^2 + ST_D)$ preprocessing, Algorithm 1 can run in $O(1)$ time per time slot. Although $T_D = \Omega(S)$ in order for the deadline to make sense, if $T_D >> \frac{S}{B_b}$ then offloading immediately would be the practical option. Therefore, we can assume that $T_D = \Theta(S)$, and the time and memory complexity of the algorithm is $O(S^2)$ in practice. Note that we were able to derive better time and memory complexity than the one implied in Section V, by taking advantage of the specific Markovian process structure of Gilbert-Elliot channels. In order to achieve similar gains for other Markovian channels, one will need to tailor the Dynamic Programming approach above to the specific structure of the channel Markov chain, if at all possible.

### A. Cloud Execution and Downloading

As stated in Section III-B, the above development was presented by taking into account only the random job uploading process. These results are easily extended to include both the (deterministic) cloud execution, i.e., $T_{exec}$ and a Markovian random downlink channel, i.e., $T_{off} = T_{up} + T_{exec} + T_{down}$ and $S = S_{up} + S_{down}$. This is done as follows. The TDAMC of Figure 2, which models the uploading of $S_{up}$ bits, is extended by branching out from each (previously) absorbing state for $T_{exec}$ transition steps. This is followed by branching out according to a process similar to the TDAMC of Figure 2, which then models the downloading of $S_{down}$ bits. The resulting Markov process therefore tracks the channel throughout all three offloading periods, i.e., upload, remote execution, and download, shown in Figure 1. The definitions of $E_{off}, E_L$, as well as the calculations carried out in Sections V-VII are then extended accordingly.

$$\hat{P}_t(T_{off},b,X_t) = \begin{cases} \sum_{k=0}^{\min(b-1,T_{off}-b-2)} \binom{b-1}{k}\binom{T_{off}-b-1}{k+1} P_{GB}^{k+1}P_{BG}^{k+1}P_{BB}^{b-k-1}P_{GG}^{T_{off}-b-k-2} & b>0 \\ P_{GG}^{T_{off}-1} & b=0 \end{cases} \quad (21)$$

$$\hat{P}_t(T_{off},b,X_t) = \begin{cases} \sum_{k=0}^{\min(b-1,T_{off}-b-2)} \binom{b-1}{k}\binom{T_{off}-b-1}{k+1} P_{GB}^{k+1}P_{BG}^{k+1}P_{BB}^{b-k-1}P_{GG}^{T_{off}-b-k-2} \\ + \sum_{k=0}^{\min(b-1,T_{off}-b-1)} \binom{b-1}{k}\binom{T_{off}-b-1}{k} P_{GB}^{k+1}P_{BG}^{k}P_{BB}^{b-k-1}P_{GG}^{T_{off}-b-k-1} & b>0 \\ P_{GG}^{T_{off}-1} & b=0 \end{cases} \quad (22)$$

$$\hat{P}_t(T_{off},b,X_t) = \sum_{k=0}^{\min(b-1,T_{off}-b-2)} \binom{b-1}{k}\binom{T_{off}-b-1}{k} P_{GB}^{k}P_{BG}^{k+1}P_{BB}^{b-k-1}P_{GG}^{T_{off}-b-k-1} \quad (23)$$

$$\hat{P}_t(T_{off},b,X_t) = \sum_{k=0}^{\min(b-1,T_{off}-b-1)} \binom{b-1}{k}\binom{T_{off}-b-1}{k} P_{GB}^{k}P_{BG}^{k+1}P_{BB}^{b-k-1}P_{GG}^{T_{off}-b-k-1} + \sum_{k=1}^{\min(b-1,T_{off}-b)} \binom{b-1}{k}\binom{T_{off}-b-1}{k-1} P_{GB}^{k}P_{BG}^{k}P_{BB}^{b-k-1}P_{GG}^{T_{off}-b-k} \quad (24)$$

## VIII. Simulation Results

In this section, computer simulation is used to study the performance of the proposed OnOpt Algorithm. As discussed in Section VII, a Gilbert-Elliot channel is assumed when offloading. It should be emphasized that based on the described system model, the optimality of the proposed OnOpt algorithm has been theoretically proved in terms of minimizing the mean energy consumption. The Gilbert-Elliott channel model is commonly used to model the effects of harsh channel conditions where burst noise can abruptly affect the data rate. The simulation results based on this channel model are used to illustrate that, even with its harsh channel conditions, there is significant gain in using the OnOpt algorithm over other heuristics. We also assume that transmit power control is used on the downlink, and therefore, $T_{down}$ (and $T_{exec}$) are deterministic. Their effects can therefore be accounted for by modifying the remote offload end-times used in the analysis. For comparison, we also plot the offline bound given in Section IV, *Local Execution* and two other algorithms, referred to as *Immediate Offloading* and *Channel Threshold*. The Local Execution algorithm executes the entire job locally without doing any offloading. For the Immediate Offloading algorithm, offloading starts at the job release time unless $S/B_g > t_D$, i.e., if offloading cannot be completed before the job deadline even with contiguous best wireless channel states, then the job is only executed locally. For the Channel Threshold algorithm, offloading starts at the first time slot when the channel condition is above a given threshold unless the remaining time before the job completion deadline is less than $S/B_g$. For the Gilbert-Elliot channel used in our results, any threshold between the good and bad states can be used, i.e., offloading starts at the first good channel time slot provided that the remaining time before the job completion deadline is no less than $S/B_g$. In both the Immediate Offloading and Channel Threshold algorithms, local execution starts at time slot $t_L$ if offloading is not completed at time slot $t_L-1$, i.e., they ensure that the job deadline is satisfied.

In the results, there are three sets of simulations, which span a wide range of parameter values. This was done to assess the relative performance of the offloading algorithms in widely varying situations. The default parameters used in the simulations are given as follows. Each time slot is taken to be 1 msec. 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 transmit power is 1 W, which means that the transmission energy for each time slot is $E_{tr} = 1$mJ. The local execution energy per CPU cycle is $v_l = 2 \times 10^{-6}$mJ and the local computation power $f_l = 1$M CPU cycles per time slot [47], [48]. We consider a job with $S = 60$Kb, $D = 10$M CPU cycles, and $t_D = 60$ time slots, where $D$ is the number of local CPU cycles needed in order to execute the job. 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. Based on $B_g$ and $B_g$, a minimum of 6 time slots and a maximum of 60 time slots are needed in order to complete job offloading. In all of the graphs, each value of average energy consumption is obtained after repeating the simulation for 10,000 runs.

### A. Scenario 1

Here we set $P_{BB} = 1 - P_{GG}$ for the channel state transition probabilities. In this case, $P_{GB} = P_{BB}$, $P_{BG} = P_{GG}$, and the equilibrium channel state probabilities are given by $P_g = P_{GG}$ and $P_b = P_{BB}$. $P_{GG}$ can therefore be used as a measure of the average channel quality. In this set we present graphs by varying parameters such as $T_D$, $S$, and good/bad state residency times.

Figure 3 shows the average energy consumption versus $T_G$, the asymptotic channel residence time in the good state, where $T_G = \frac{1}{P_{GB}}$. The energy used by Local Execution is obviously constant for all residence times. When the good state residence time is low, the OnOpt algorithm does not offload because there is not enough time to complete the offload, or, the

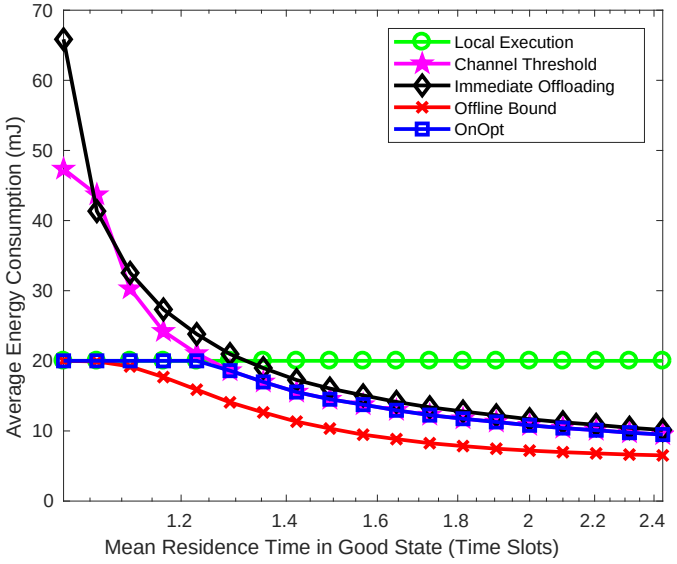


Fig. 3. Average energy consumption versus channel residence time in good state

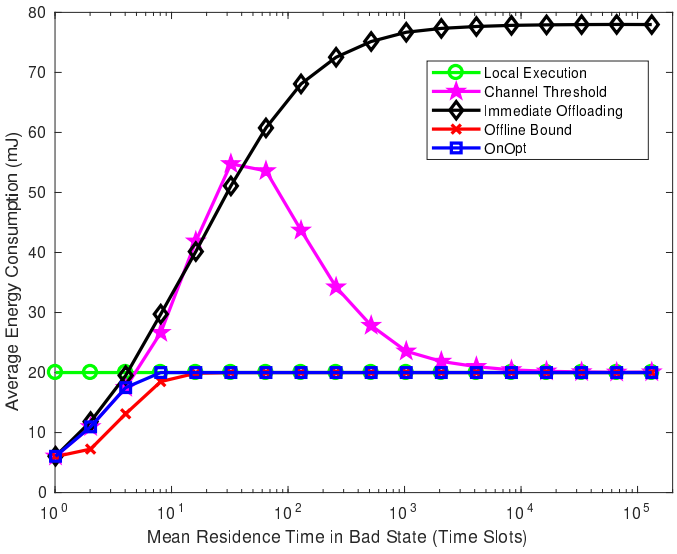


Fig. 4. Average energy consumption versus channel residence time in bad state

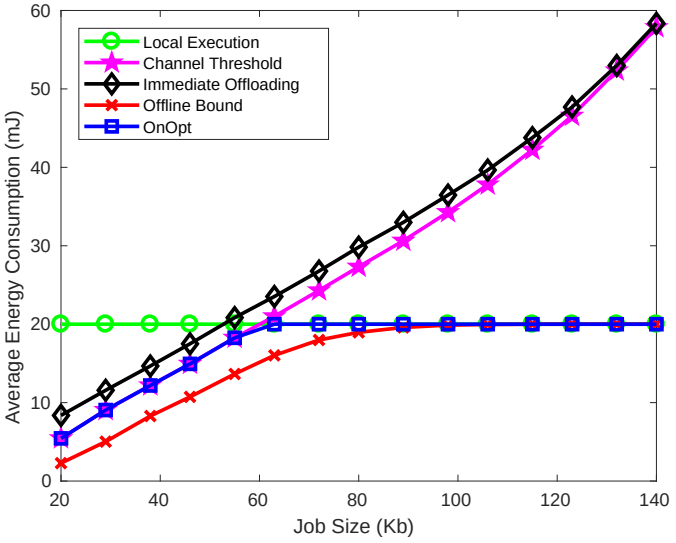


Fig. 5. Average energy consumption versus data size $S$: $P_{GG} = 0.2$

expected energy is higher than $E_L$. As the residence time increases, the energy consumption for OnOpt decreases. The energy consumption for Channel Threshold and Immediate Offloading decreases as the residence time in the good state increases. The energy for these algorithms is above $E_L$ when the residence time is low.

Figure 4 shows the average energy consumption versus $T_B$, the asymptotic mean channel residence time in the bad state, where $T_B = \frac{1}{P_{BG}}$. Figure 4 shows that as the bad state residence time increases, the energy consumption for all of the algorithms initially increases. When $T_B$ is above about 10 time slots, both the offline bound and the OnOpt algorithm do not offload due to the long time needed, eventually resulting in the same energy consumption as Local Execution. For the Channel Threshold algorithm, as $T_B$ increases, offloading may still be possible either because the channel is in the good state at the release time or the first good channel state appears not long afterwards. However, the probability that the offload can be completed before $t_D$ decreases as $T_B$ increases. Therefore, the energy consumption increases with $T_B$. As $T_B$ further increases, offloading is possible only if the channel is in the good state at the release time (the probability decreases as $T_B$ increases), and therefore, the energy consumption decreases. In Immediate Offloading, the energy consumption increases with $T_B$ until $T_B$ is so large that the channel is practically always in the bad state. The energy consumption, in this case, converges to $E_L + E_{tr}S/B_b = 80$mJ.

Figure 5 shows the average energy consumption of the mobile device as the data size $S$ increases. When $S$ is small, offloading can most likely meet the delay constraint without local execution. The average energy consumption of the Channel Threshold and OnOpt algorithms is the same, since the two algorithms offload at the same time slot, while the Immediate Offloading algorithm consumes higher energy for the same reason as explained previously. As $S$ increases, a longer time is needed for wireless transmission, and both the offline and OnOpt algorithms may decide not to offload, resulting in the same energy consumption as Local Execution, while the Immediate Offloading and Channel Threshold algorithms waste energy by offloading unnecessarily, which results in much higher energy consumption.

Figure 6 shows the average energy consumption versus the local execution duration time. Here, we change $D$ from 2 to 15 CPU Mega-cycles. Deadline $T_D$ is set to 20 time slots in order to make the deadline tighter and observe the effects of increasing $T_L$. When $E_L$ is small, the OnOpt algorithm does not offload because the expected cost is higher than $E_L$. When $E_L$ becomes large enough, the OnOpt algorithm starts offloading, thus reducing its energy use. Increasing $T_L$ increases the chance that overlap occurs between local execution and offloading. Therefore, the energy consumption for OnOpt starts to increase. A similar situation happens for the other algorithms.

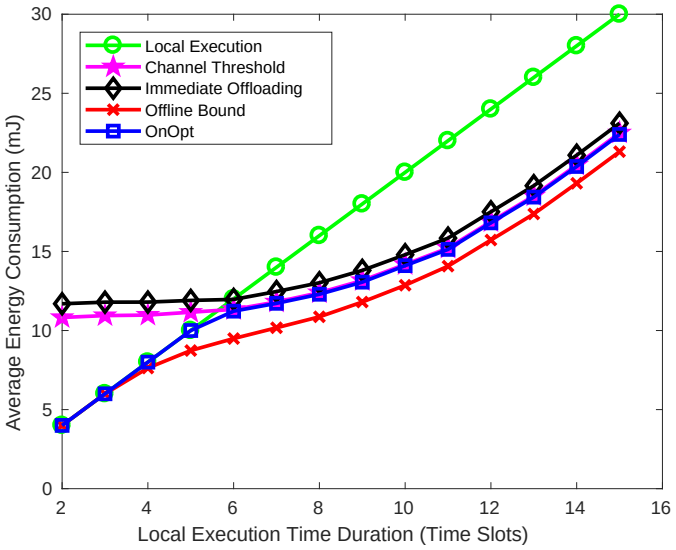


Fig. 6. Average energy consumption versus local execution time $T_L$: $P_{GG} = 0.5$

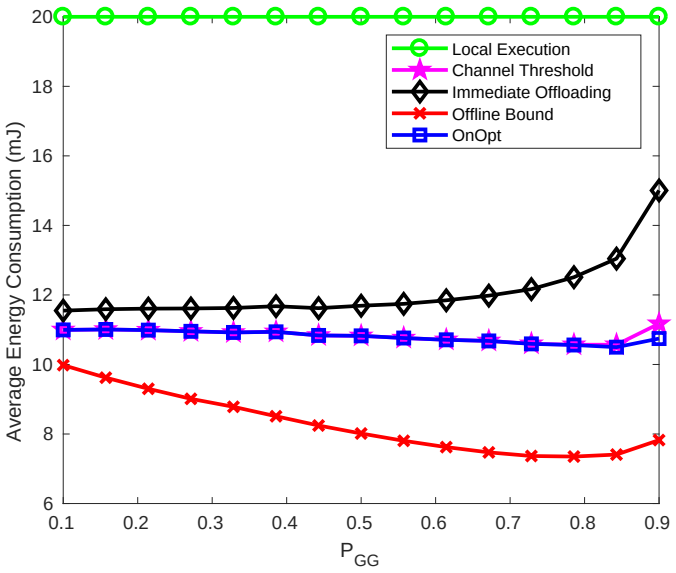


Fig. 7. Average energy consumption versus $P_{GG}$: $t_D = 40$ time slots

## B. Scenario 2

In the second set, we set $P_{BB} = P_{GG}$, so that the equilibrium channel state probabilities are equal, but by varying these parameters, we can observe the effects of mean channel state residency time. The channel state transition probabilities are assumed to satisfy $P_{BB} = P_{GG}$. In this case, the equilibrium channel state probabilities are equal, and therefore, a larger $P_{GG}$ does not indicate better channel quality on average. Instead, it represents how dynamically the channel state changes. When $P_{GG}$ (and $P_{BB}$) is large for example, once the channel enters a particular state, it is more likely to persist in that state, i.e., more consecutive time slots in the same state are likely. The opposite is true when $P_{GG}$ (and $P_{BB}$) are made smaller. By varying $P_{GG}$, the average energy consumption of all four algorithms are given in Figure 7 for $t_D = 40$ time slots and Figure 8 for $t_D = 20$ time slots.

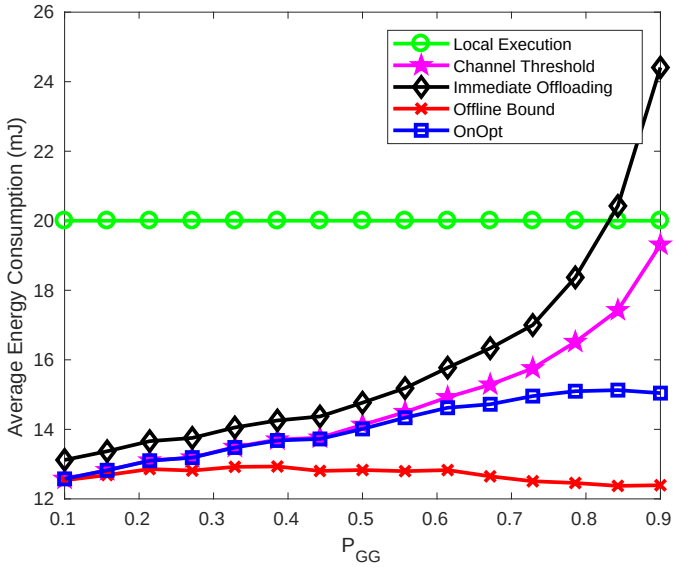


Fig. 8. Average energy consumption versus $P_{GG}$: $t_D = 20$ time slots

The offline solution can foresee future channel states, and a larger $P_{GG}$ makes it more likely to choose consecutive time slots with good channel states. Therefore, the average energy consumption of the offline bound decreases as $P_{GG}$ increases. When $P_{GG}$ is very close to zero, the channel state is likely to toggle in the next time slot. In this case, the Immediate Offloading algorithm consumes about 0.5mJ extra energy, compared to the Channel Threshold algorithm, i.e., 0.5mJ is 50% (which is the probability that the channel state at the job release time is bad) times 1mJ (which is the transmission energy in the first time slot). As $P_{GG}$ increases, it is increasingly likely to have consecutive time slots with the same channel conditions. If the channel is in the good state when a job is released, the Immediate Offloading and Channel Threshold algorithms are the same. However, if the channel is in the bad state when a job is released, it is likely that the bad channel state persists for a relatively long time, during which Immediate Offloading may waste energy. Therefore, with higher $P_{GG}$, the difference between Immediate Offloading and the Channel Threshold algorithm increases. The OnOpt and the Channel Threshold algorithms are very close when $t_D = 80$ time slots since the time constraint is loose enough for the OnOpt algorithm to offload at the first time slot with good channel conditions. When $t_D = 35$ time slots, the difference between the two algorithms starts increasing as $P_{GG}$ becomes large. This is because OnOpt has the flexibility to offload at a bad time slot while the Channel Threshold algorithm does not. As a result, the OnOpt may finish offloading much sooner than the Channel Threshold algorithm.

Figure 9 shows the average energy consumption versus $T_G$, which is the asymptotic average channel residence time in the good state, where $T_G = 1/P_{GB} = 1/(1 - P_{GG})$. Note that in this set of results, $T_B = T_G$ since $P_{GG} = P_{BB}$. When $T_G$ is below about 10 time slots (i.e., $P_{GG}$ is between 0.1 and 0.9), the observations are the same as seen in Figure 7. Therefore, the discussion below is only for $T_G > 10$ time

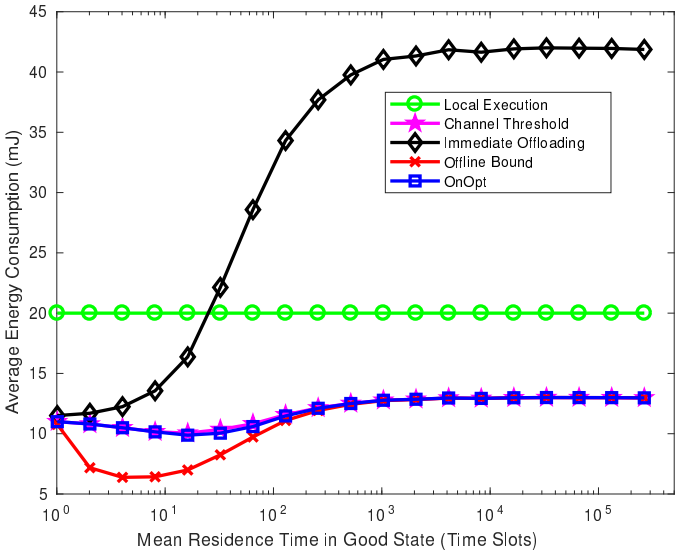


Fig. 9. Average energy consumption versus residency time in good state: $t_D = 40$ time slots

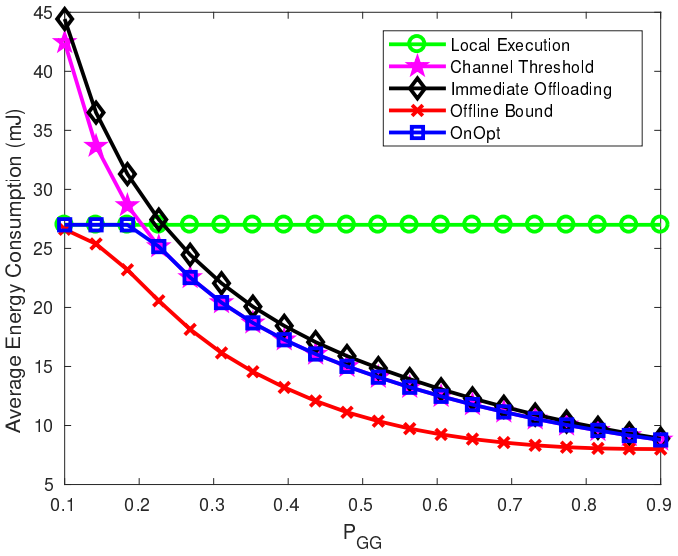


Fig. 10. Average energy consumption versus $P_{GG}$

slots. The Immediate Offloading algorithm can consume much higher energy than the others, since it may have to transmit for a long time if the channel is in the bad state at the job release time. The OpOpt and Channel Threshold algorithms are essentially the same, since both decide to offload at the first time slot with the good channel state.

### C. Scenario 3

In this set of results, we use the application parameters for x264 (H.264) encoding from [49], and consider a job with $S = 80$Kb, $D = 18$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. Based on $B_g$ and $B_g$, a minimum of 8 time slots and a maximum of 80 time slots are needed in order to complete job offloading. In addition to the results presented below, we have also simulated the algorithms based on parameters given in [50]. This reference does experiments of computation offloading for face recognition on mobile devices. Since the qualitative observations and conclusions are the same as those presented below, these results have not been included.

In this case we set $P_{BB} = 1 - P_{GG}$ for the channel state transition probabilities. As discussed previously, $P_{GG}$ is a measure of the average channel quality. Figure 10 shows the average energy consumption of different algorithms as $P_{GG}$ is varied. The offline bound is the same as the energy consumption of Local Execution only when $P_{GG}$ is close to 0 and it decreases as $P_{GG}$ increases. When $P_{GG}$ is very low, the offline optimal solution chooses to process the job locally without offloading because of the long data transmission time (and possibly a long overlap time between offloading and local execution). As a result, there is a high probability that offloading cannot meet the delay constraint and/or consumes higher energy than $E_L$. As $P_{GG}$ becomes larger, the good channel state frequency increases, and a shorter time is needed to complete the offloading. In this case, it is more likely that offloading can meet the delay constraint and consume less energy. The Immediate Offloading algorithm results in much higher energy consumption when $P_{GG}$ is small. Since the channel is in the bad state in most time slots, it is less likely that offloading can meet the deadline, and the deadline of the job is most likely met by performing local execution. Therefore, energy is unnecessarily wasted 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. The Channel Threshold algorithm consumes slightly lower energy than Immediate Offloading. By delaying the offloading until the channel is in the good state, unnecessary transmissions are avoided. The difference is more obvious when $P_{GG}$ is smaller, since the probability is higher that the channel is found in the bad state. For the proposed OnOpt algorithm, it chooses to not offload when $P_{GG}$ is low, and therefore, results in the same energy consumption as Local Execution. When $P_{GG}$ is larger, channel conditions become better and a shorter time is needed to offload. Given that the offloading decision is made using only the current channel state and statistical channel information, if the decision is to offload at a time slot, it is most likely the first time slot with a good channel state. Therefore, the OnOpt and Channel Threshold algorithms consume almost the same energy when $P_{GG}$ is relatively large. The gap between the OnOpt algorithm and the offline bound is due to the fact that the online algorithm can only use statistical channel information, while the offline bound has knowledge of future channel conditions.

Figure 11 shows the average energy consumption of the algorithms as the job deadline $t_D$ changes. 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

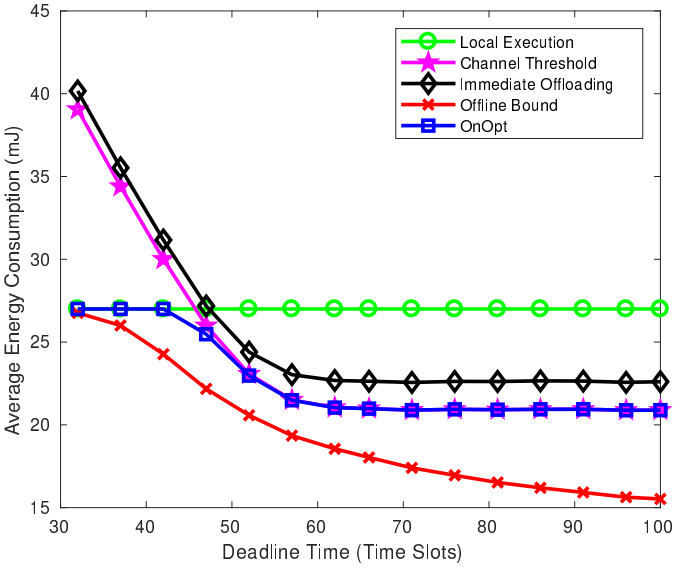


Fig. 11. Average energy consumption versus $t_D$: $P_{GG} = 0.3$

transmissions. Ideally, the minimum energy consumption is achieved if eight consecutive time slots with good channel states appear before $t_L$. The probability of this decreases as the deadline is tightened. However, a larger $t_D$ increases the possibility of finding a shorter time interval to complete the offloading, thus reducing the energy consumption. When $t_D$ is sufficiently large, it is almost always possible to find consecutive time slots with good channel states, and therefore, increasing $t_D$ further cannot significantly decrease the average energy consumption. The Channel Threshold and OnOpt algorithms result in similar average energy consumption, which is slightly lower than using Immediate Offloading and much lower than using Local Execution, provided that $t_D$ is not too small. As $t_D$ increases, more time is available to offload before triggering local execution, resulting in lower energy consumption. When $t_D$ is sufficiently large, Channel Threshold and OnOpt all start offloading at the first time slot with a good channel state, while Immediate Offloading may have to transmit over an initial bad channel, resulting in slightly higher energy consumption than the other two algorithms. When $t_D$ is sufficiently large so that offloading can always be completed before $t_L$ regardless of the initial channel state, further increasing $t_D$ does not help in reducing the energy consumption. This is true for all three online algorithms.

## IX. Conclusions

This paper has considered the mobile computation offloading case where job completion times are subject to hard deadline constraints. Instead of using conventional offload/no-offload execution decisions, the paper allows simultaneous remote offloading and local job execution, which is used to ensure that job completion deadlines are met in the face of random channel conditions. The paper considered this problem when the wireless channels are modelled as homogeneous Markovian processes. The OnOpt (Online Optimum) algorithm was proposed that achieves the minimum mean energy consumption possible. This was shown by first constructing a time-dilated absorbing Markov chain (TDAMC) from the underlying Markov channel description. Dynamic programming results were then used with the TDAMC to show OnOpt's optimality. This resulted in a simple test that can be performed to determine if the current time is best for initiating a computation offload. The paper then used the Gilbert-Elliott channel model and derived closed-form results that are used to find optimal offload initiation times. The job completion time probabilities were computed recursively, which leads to a a large reduction in the computational complexity. The performance of the proposed algorithm was compared to three others that also ensure that job deadlines are satisfied, i.e., Immediate Offloading, Channel Threshold, and Local Execution. An offline lower bound on energy consumption was computed and used in these comparisons. Performance results show that the proposed algorithm can significantly improve mobile device energy consumption compared to the other approaches while guaranteeing hard task execution deadlines. When the channel conditions are relatively poor or the time constraint is relatively tight, the proposed algorithm may decide to not offload, saving energy by not transmitting unnecessarily. When the channel conditions are good on average, the proposed algorithm can choose the earliest transmission time, saving as much energy for local execution as possible or not triggering local execution at all. Overall, the advantage of the proposed algorithm has been demonstrated over a wide range of parameter values.

## References

[1] "Cisco Visual Networking Index: Forecast and Methodology, 2016-2021," *Cisco, San Jose, CA, USA*, 2017.

[2] M. Satyanarayanan, "Fundamental challenges in mobile computing," in *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*. ACM, 1996, pp. 1–7.

[3] 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.

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

[5] 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.

[6] ETSI, "Mobile-edge computing introductory technical white paper," in *[Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge computing - introductory technical white paper v1*, September 2014.

[7] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2322–2358, August 2017.

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

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

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

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

[12] 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.

[13] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving Portable Computer Battery Power through Remote Process Execution," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 2, no. 1, pp. 19–26, January 1998.

[14] ——, "The remote processing framework for portable computer power saving," in *Proceedings of the 1999 ACM symposium on Applied computing*, March 1999, pp. 365–372.

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

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

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

[18] R. Kaewpuang, D. Niyato, P. Wang, and E. Hossain, "A Framework for Cooperative Resource Management in Mobile Cloud Computing," *IEEE Journal on Selected Areas in Communications*, vol. 31, no. 12, pp. 2685–2700, December 2013.

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

[20] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint Optimization of Radio and Computational Resources for Multicell Mobile-Edge Computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, June 2015.

[21] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[22] S. Kosta, A. Andrius, P. Hui, R. Mortier, and 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)*, March 2012, pp. 945–953.

[23] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, May 2015, pp. 1894–1902.

[24] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading decision and resource allocation for multi-user multi-task mobile cloud," in *Proceedings of IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[25] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy Aware Offloading for Competing Users on a Shared Communication Channel," *IEEE Transactions on Mobile Computing*, vol. 16, no. 1, pp. 87–96, January 2017.

[26] ——, "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.

[27] S. Joilo and G. Dn, "A game theoretic analysis of selfish mobile computation offloading," in *Proceedings of IEEE International Conference on Computer Communications (INFOCOM)*, May 2017, pp. 1–9.

[28] H. Cao and J. Cai, "Distributed Multiuser Computation Offloading for Cloudlet-Based Mobile Cloud Computing: A Game-Theoretic Machine Learning Approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 752–764, January 2018.

[29] 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.

[30] E. N. Gillbert, "Capacity of a burst noise channel," *Bell Syst. Tech. J.*, vol. 39, pp. 1253–1266, September 1960.

[31] E. O. Elliott, "Estimates of error rates for codes on burst-noise channels," *Bell Syst. Tech. J.*, vol. 42, pp. 1977–1997, September 1963.

[32] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya, "Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 337–368, 2014.

[33] A. u. Rehman Khan, M. Othman, S. A. Madani, and S. Ullah Khan, "A Survey of Mobile Cloud Computing Application Models," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 393–413, 2014.

[34] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, "Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 14–22, 2013.

[35] L. Guan, X. Ke, M. Song, and J. Song, "A Survey of Research on Mobile Cloud Computing," in *IEEE/ACIS International Conference on Computer and Information Science*, May 2011, pp. 387–392.

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

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

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

[39] Z. Guan and T. Melodia, "Cloud-Assisted Smart Camera Networks for Energy-Efficient 3D Video Streaming," *Computer*, vol. 47, no. 5, pp. 60–66, 2014.

[40] ——, "The Value of Cooperation: Minimizing User Costs in Multi-Broker Mobile Cloud Computing Networks," *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 780–791, 2015.

[41] Y. Geng, Y. Yang, and G. Cao, "Energy-Efficient Computation Offloading for Multicore-Based Mobile Devices," in *IEEE Conference on Computer Communications (INFOCOM)*, October 2018, pp. 46–54.

[42] M. Zafer and E. Modiano, "Minimum energy transmission over a wireless fading channel with packet deadlines," in *Proceedings of IEEE International Conference on Decision and Control*, December 2007, pp. 1148–1155.

[43] L. A. Johnston and V. Krishnamurthy, "Opportunistic file transfer over a fading channel: A POMDP search theory formulation with optimal threshold policies," *IEEE Transactions on Wireless Communications*, vol. 5, no. 2, pp. 394–405, February 2006.

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

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

[46] M. Zed, R. R. Rao, and L. B. Milstein, "On the accuracy of a first-order Markov model for data transmission on fading channels," in *IEEE International Conference on Universal Personal Communications*, November 1995, pp. 211–215.

[47] 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.

[48] 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.

[49] A. P. Miettinen and 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.

[50] N. Sumi, A. Baba, and V. G. Moshnyaga, "Effect of computation offload on performance and energy consumption of mobile face recognition," in *IEEE Workshop on Signal Processing Systems (SiPS)*, Oct 2014, pp. 1–7.